(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
23 August 2001 (23.08.2001)

PCT

(10) International Publication Number
WO 01/61703 A2

(51) International Patent Classification[7]: G11C

(21) International Application Number: PCT/US01/05052

(22) International Filing Date: 13 February 2001 (13.02.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/505,555 17 February 2000 (17.02.2000) US

(71) Applicant: SANDISK CORPORATION [US/US]; 140 Caspian Court, Sunnyvale, CA 94089 (US).

(72) Inventors: CONLEY, Kevin, M.; 5983 Alvarado Court, San Jose, CA 95120 (US). MANGAN, John, S.; 2158 Sunny Acres Drive, Santa Cruz, CA 95060 (US). CRAIG, Jeffrey, G.; 44030 Geddy Court, Fremont, CA 94539 (US).

(74) Agent: PARSONS, Gerald, P.; Skjerven Morrill MacPherson LLP, Three Embarcadero Center, 28th Floor, San Francisco, CA 94111 (US).

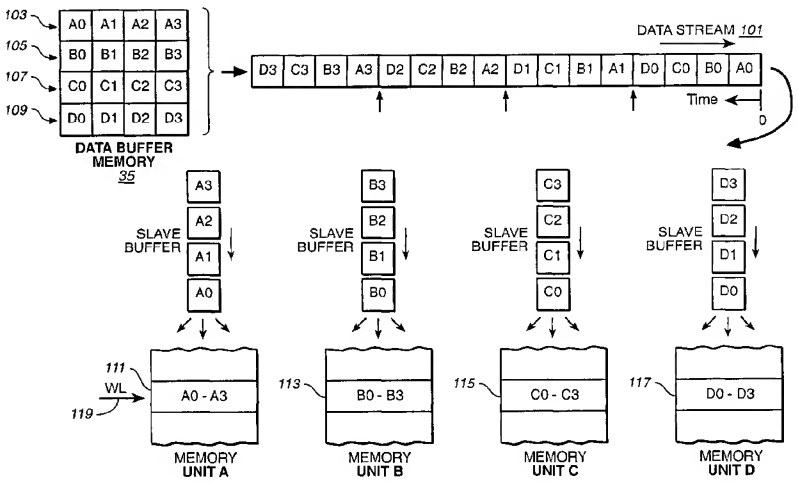
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

[Continued on next page]

(54) Title: FLASH EEPROM SYSTEM WITH SIMULTANEOUS MULTIPLE DATA SECTOR PROGRAMMING AND STORAGE OF PHYSICAL BLOCK CHARACTERISTICS IN OTHER DESIGNATED BLOCKS



(57) Abstract: A non-volatile memory system is formed of floating gate memory cells arranged in blocks as the smallest unit of memory cells that are erasable together. The system includes a number of features that may be implemented individually or in various cooperative combinations. One feature is the storage in separate blocks of the characteristics of a large number of blocks of cells in which user data is stored. These characteristics for user data blocks being accessed may, during operation of the memory system by its controller, be stored in a random access memory for ease of access and updating. According to another feature, multiple sectors of user data are stored at one time by alternately streaming chunks of data from the sectors to multiple memory blocks. Bytes of data in the stream may be shifted to avoid defective locations in the memory such as bad columns. Error correction codes may also be generated from the streaming data with a single generation circuit for the multiple sectors of data. The stream of data may further be transformed in order to tend to even out the wear among the blocks of memory. Yet another feature, for memory systems having multiple memory integrated circuit chips, provides a single system record that includes the capacity of each of the chips and assigned contiguous logical address ranges of user data blocks within the chips which the memory controller accesses when addressing a block, making it easier to manufacture a memory system with memory chips having different capacities. A typical form of the memory system is as a card that is removably connectable with a host system but may alternatively be implemented in a memory embedded in a host system. The memory cells may be operated with multiple states in order to store more than one bit of data per cell.

WO 01/61703 A2

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

FLASH EEPROM SYSTEM WITH SIMULTANEOUS MULTIPLE DATA SECTOR PROGRAMMING AND STORAGE OF PHYSICAL BLOCK CHARACTERISTICS IN OTHER DESIGNATED BLOCKS

BACKGROUND OF THE INVENTION

This invention relates to semiconductor memory systems, particularly to non-volatile memory systems, and have application to flash electrically-erasable and programmable read-only memories (EEPROMs).

Flash EEPROM systems are being applied to a number of applications, particularly when packaged in an enclosed card that is removably connected with a host system. Current commercial memory card formats include that of the Personal Computer Memory Card International Association (PCMCIA), CompactFlash (CF), MultiMediaCard (MMC) and Secure Digital (SD). One supplier of these cards is SanDisk Corporation, assignee of this application. Host systems with which such cards are used include personal computers, notebook computers, hand held computing devices, cameras, audio reproducing devices, and the like. Flash EEPROM systems are also utilized as bulk mass storage embedded in host systems.

Such non-volatile memory systems include an array of floating-gate memory cells and a system controller. The controller manages communication with the host system and operation of the memory cell array to store and retrieve user data. The memory cells are grouped together into blocks of cells, a block of cells being the smallest grouping of cells that are simultaneously erasable. Prior to writing data into one or more blocks of cells, those blocks of cells are erased. User data are typically transferred between the host and memory array in sectors. A sector of user data can be any amount that is convenient to handle, preferably less than the capacity of the memory block, often being equal to the standard disk drive sector size, 512 bytes. In one commercial architecture, the memory system block is sized to store one sector of user data plus overhead data, the overhead data including information such as an error correction code (ECC) for the user data stored in the block, a history of use of the block, defects and other physical information of the memory cell block. Various implementations of this type of non-volatile memory system are described in the

following United States patents and pending applications assigned to SanDisk Corporation, each of which is incorporated herein in its entirety by this reference: Patents nos. 5,172,338, 5,602,987, 5,315,541, 5,200,959, 5,270,979, 5,428,621, 5,663,901, 5,532,962, 5,430,859 and 5,712,180, and application serial nos. 08/910,947, filed August 7, 1997, and 09/343,328, filed June 30, 1999. Another type of non-volatile memory system utilizes a larger memory cell block size that stores multiple sectors of user data.

One architecture of the memory cell array conveniently forms a block from one or two rows of memory cells that are within a sub-array or other unit of cells and which share a common erase gate. United States patents nos. 5,677,872 and 5,712,179 of SanDisk Corporation, which are incorporated herein in their entirety, give examples of this architecture. Although it is currently most common to store one bit of data in each floating gate cell by defining only two programmed threshold levels, the trend is to store more than one bit of data in each cell by establishing more than two floating-gate transistor threshold ranges. A memory system that stores two bits of data per floating gate (four threshold level ranges or states) is currently available, with three bits per cell (eight threshold level ranges or states) and four bits per cell (sixteen threshold level ranges) being contemplated for future systems. Of course, the number of memory cells required to store a sector of data goes down as the number of bits stored in each cell goes up. This trend, combined with a scaling of the array resulting from improvements in cell structure and general semiconductor processing, makes it practical to form a memory cell block in a segmented portion of a row of cells. The block structure can also be formed to enable selection of operation of each of the memory cells in two states (one data bit per cell) or in some multiple such as four states (two data bits per cell), as described in SanDisk Corporation United States patent no. 5,930,167, which is incorporated herein in its entirety by this reference.

Since the programming of data into floating-gate memory cells can take significant amounts of time, a large number of memory cells in a row are typically programmed at the same time. But increases in this parallelism causes increased power requirements and potential disturbances of charges of adjacent cells or interaction between them. United States patent no. 5,890,192 of SanDisk Corporation, which is incorporated herein in its entirety, describes a system that

minimizes these effects by simultaneously programming multiple chunks of data into different blocks of cells located in different operational memory cell units (sub-arrays).

SUMMARY OF THE INVENTION

There are several different aspects of the present invention that provide improvements in solid state memory systems, including those described above. Each of these aspects of the present invention, the major ones being generally and briefly summarized in the following paragraphs, may be implemented individually or in various combinations.

Multiple user data sectors are programmed into a like number of memory blocks located in different units or sub-arrays of the memory array by alternately streaming data from one of the multiple sectors at a time into the array until a chunk of data is accumulated for each of multiple data sectors, after which the chunks are simultaneously and individually stored in respective blocks in different units of the memory. This increases the number of memory cells that may be programmed in parallel without adverse effects.

An error correction code (ECC), or other type of redundancy code, may be generated by the controller from the streaming user data during programming and written into the same memory block as the user data from which it is generated. The redundancy code is then evaluated by the controller when the sector of data is read out of the memory block. A single redundancy code generation circuit is utilized, even when the streaming data is alternated between data chunks of the multiple sectors, by providing a separate storage element for each of the user data sectors being programmed at the same time, in which intermediate results of the generation are temporarily stored for each sector.

Overhead data of the condition, characteristics, status, and the like, of the individual blocks are stored together in other blocks provided in the array for this purpose. Each overhead data record may include an indication of how many times the block has been programmed and erased, voltage levels to be used for programming and/or erasing the block, whether the block is defective or not, and, if so, an address of a substitute good block, and the like. A group of blocks are devoted to storing such records. A large number of such records are stored in each of these overhead blocks.

When accessing a specific user data block to perform one or all of programming, reading or erasing, the overhead record for that user data block is first read and its information used in accessing the block. By storing a block's overhead data outside of that block, frequent rewriting of the overhead data, each time the user data is rewritten into the block, is avoided. It also reduces the amount of time necessary to access and read the block overhead data when the block is being accessed to read or write user data. Further, only one ECC, or other redundancy code, need be generated for the large number of overhead records that are stored in this way.

The records from a number of overhead blocks can be read by the controller into an available portion of its random-access memory for ease of use, with those overhead blocks whose records have not been accessed for a time being replaced by more active overhead blocks in a cache-like manner. When a beginning address and number of sectors of data to be transferred is received by the memory controller from the host system, a logical address of the first memory block which is to be accessed is calculated in order to access the overhead record for that block but thereafter the overhead records are accessed in order without having to make a calculation of each of their addresses. This increases the speed of accessing a number of blocks. Information of defects in the memory, such as those discovered during the manufacturing process, may also be stored in separate blocks devoted for this purpose and used by the controller so that the imperfect memory circuit chips may be included in the memory system rather than discarding them. This is particularly an advantage when a single defect record affects many blocks. One such defect is a bad column that is shared by a large number of blocks. A number of bad column pointers (BCPs) may be stored together as a table in one or more sectors that are devoted in part or entirely to this overhead data. When this is done, the physical location of the streaming user data being written to the memory is shifted when a comparison of the relative physical location within a sector of individual bytes of data with the BCP table indicates that the data byte is being directed to at least one memory cell that is along a bad column. The reverse is performed during reading, where data bits read from memory cells that were skipped over during write because of a bad column are ignored.

Since flash EEPROM cells have, by their nature, a limited life in terms of the number of times that they can be erased and reprogrammed, it is usually prudent to

include one or more operational features that tend to even out the wear on the various memory blocks that can be caused by multiple rewrites to the same blocks. One such technique alters from time-to-time the correspondence between the digital data and the memory states that are designated to represent the digital data. To accomplish this in the present memory system, the first one or few bits of the initial byte of the individual sectors of data, termed herein as a flag byte, are used to designate such correspondence. These bits are designated upon writing user data, and all data following the initial bits are transformed on the fly as the streaming data is being transferred to the memory array in accordance with their value. Upon reading a sector of data, these initial bit(s) are read and used to transform back all subsequent data stored in the sector to their original values as the data are being read out of the memory in a stream.

When the memory system is formed of multiple memory cell arrays, such as by using two or more integrated circuit chips that each include such an array, the system's manufacture and use is simplified by accumulating information about each of the memory arrays in the system, and then storing that information as a single record in some convenient location, such as in one of the blocks of one of the memory arrays. This makes it much easier to combine memory arrays having different sizes and/or operating characteristics into a single system. One such record merges the number of blocks of user data available in each of the memory chips in a way that establishes a continuum of logical block addresses of the blocks of all the arrays in the system. When a location of memory is being accessed for a read or write operation, the memory controller then accesses the merged record in a process of converting a logical block address to a physical address of a block in one of the memory arrays.

Such a merged record can be automatically generated and stored during manufacturing by the controller reading the information from each of the memory arrays, merging that information into a single record, and then writing that record into a designated block of one of the memory arrays. Currently, the memory controller is usually provided on a separate integrated circuit chip, with one or more memory cell array chips connected with the controller. It is contemplated, in light of continuing processing technology improvements, that a memory array can also be included on the controller chip. When that amount of memory is insufficient for a particular system, one or more additional circuit chips containing further memory array(s) are

then utilized. When two or more physically separate arrays of included in a system, then the generation of the merged record simplifies operation of the controller to address blocks across multiple arrays.

Other characteristics of various memory array circuit chips used to form a system, such as optimal voltage ranges, timing, numbers of pulses used, characteristics of voltage pumps, locations of overhead blocks, and the like, can vary without adverse effects. These operating characteristics can also be tabulated into a single system file for access by the micro-controller, or, more conveniently, the micro-controller can be operated to first access those of such characteristics as are necessary from an individual memory array chip before accessing that chip to read or write data. In either case, this allows the memory system to be formed with memory chips having different characteristics without any degradation in its performance. The manufacturing of non-volatile memory systems is simplified since all the memory array chips in a system need not be the selected to be the same.

Additional aspects, features and advantages of the present invention are included in the following description of specific embodiments, which description should be taken in conjunction with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a block schematic diagram of an example non-volatile memory system in which various aspects of the present invention may be implemented;

Figure 2 is a more detailed schematic diagram of one of the memory cell array units, with associated logic and buffers, of the system of Figure 1;

Figures 3A and 3B illustrate four state and two state operation, respectively, of the individual memory cells of the system of Figure 1;

Figure 4 is an example of the content and structure of data stored in one of the memory blocks of the system of Figure 1;

Figure 5 illustrates streaming data transfer during programming of multiple sectors of data at one time within the memory system of Figure 1;

Figure 6A is a schematic block diagram of the circuit of the system of Figure 1 that utilizes bad column pointers (BCP) and generates an error correction code (ECC), or other data redundancy code, with the multi-sector streaming data being transferred to the memory cell array;

Figure 6B is a schematic diagram of the ECC generation block of Figure 6A;

Figure 6C is a schematic diagram of the BCP processing block of Figure 6A;

Figure 7 illustrates the manner in which data from multiple sectors is transferred to the memory cell array;

Figure 8 illustrates an example data structure of block overhead data records;

Figure 9 shows an example of an overhead data record of Figure 8 for a good memory cell block;

Figure 10 shows an example of an overhead data record of Figure 8 for a defective memory cell block;

Figure 11 illustrates an example data structure of a reserved block that stores records of defects of the memory cell array, namely the location of bad columns, of the system of Figure 1;

Figure 12 shows an example of the physical partitioning of the memory array of Figure 1 into units and blocks of memory cells, and the data that is designated to be stored in them;

Figure 13 schematically illustrates a method of forming a merged record of the characteristics of multiple memory integrated circuit chips of the system of Figure 1;

Figure 14 shows an example of a merged record formed in the manner illustrated in Figure 13;

Figure 15 illustrates an example of bit fields of the flag byte of the data sector of Figure 4;

Figure 16 schematically shows generation and use of the bit fields of the flag byte of Figure 15 during writing of data in the memory; and

Figure 17 schematically shows use of the bit fields of the flag byte of Figure 15 during reading of data from the memory.

DESCRIPTION OF REPRESENTATIVE EMBODIMENTS

Figure 1 provides a diagram of the major components of a non-volatile memory system that are relevant to the present invention. A controller 11 communicates with a host system over lines 13. The controller 11, illustrated to occupy one integrated circuit chip, communicates over lines 15 to one or more non-volatile memory cell arrays, three arrays 17, 19 and 21 being illustrated, each of the arrays usually formed on one or more separate integrated circuit chips. The illustrated

controller is usually contained on a single integrated circuit chip, either without a flash EEPROM array (the example shown) or with a memory cell array. Even if a memory cell array is included on the controller circuit chip, an additional one or more chips that each contain only a memory array and associated circuitry will often be included in the system.

User data is transferred between the controller 11 and multiple memory arrays 17, 19 and 21, in this example, over the lines 15. The memory arrays are individually addressed by the controller. Specifically, the data bus within the lines 15 can be one byte wide. The memory system shown in Figure 1 can be embedded as part of a host system or packaged into a card, such as a card following one of the card standards previously mentioned. In the case of a card, the lines 13 terminate in external terminals on the card for mating with a complementary socket within a host system. Although use of one controller chip and multiple memory chips is typical, the trend is, of course, to use fewer separate chips for such a system by combining their circuits. An example capacity of one of the illustrated memory chips is 256 Mbits, thus requiring only two such memory chips, plus the controller chip, to form a non-volatile memory system having a data capacity of 64 megabytes. Use of a single smaller capacity memory chip results in a memory system of lesser capacity, an 8 megabyte system being a marketable example. Conversely, use of memory chips with a higher bit storage density and/or use of more memory array chips in a system will result in a higher capacity memory. Such memory systems up to 1.3 gigabyte and more are possible.

The controller 11 includes a micro-processor or micro-controller 23 connected through controller interface logic 25 to internal memories and interfaces with external components. A program memory 27 stores the firmware and software accessed by the micro-controller 23 to control the memory system operation to read data from the connected memory array(s) and transmit that data to the host, to write data from the host to the memory chip(s), and to carry out numerous other monitoring and controlling functions. The memory 27 can be a volatile re-programmable random-access-memory (RAM), a non-volatile memory that is not re-programmable (ROM), a one-time programmable memory (OTP) or a re-programmable flash EEPROM system. If the memory 27 is re-programmable, the controller can be configured to allow the host system to program it. A random-access-memory (RAM) 29 is used to

store, among other data, data from tables read from the non-volatile memory that are accessed during reading and writing operations.

A logic circuit 31 interfaces with the host communication lines 13, while another logic circuit 33 interfaces with the memory array(s) through the lines 15. Another memory 35 is used as a buffer to temporarily store user data being transferred between the host system and non-volatile memory. The memories in the controller are usually volatile, since memories with fast access and other characteristics desired for efficient controller access have that characteristic, and may be combined physically into a single memory. A dedicated circuit 36 accesses the streaming user data being transferred to the memory and inserts dummy bytes into the data stream in order to avoid writing valid user data to memory cells in bad columns. A dedicated processing circuit 37 also accesses the streaming user data being transferred between the controller and flash interfaces 25 and 33 for generating an ECC, or other type of redundancy code, based upon the user data. When user data is being transferred into the non-volatile memory, the generated ECC is appended onto the user data and simultaneously written into the same physical block of the non-volatile memory as part of the same sector as the user data. The circuits 36 and 37 are described further below, with respect to Figures 6A-C.

The non-volatile memory chip 17 includes a logic circuit 39 for interfacing with the controller through the lines 15. Additional components of the memory chip are not shown for simplicity in explanation. The purpose of the logic circuit 39 is to generate signals in separate buses and control lines. Various control signals are provided in lines 41 and a power supply 43 to the memory array circuits is also controlled through the interface 39. A data bus 45 carries user data being programmed into or read from the non-volatile memory, and an address bus 47 carries the addresses of the portion of the memory being accessed for reading user data, writing user data or erasing blocks of memory cells.

The floating gate memory cell array of a single non-volatile memory chip is itself divided into a number of units that each have its own set of supporting circuits for addressing, decoding, reading and the like. In this example, eight such array units 0-7, denoted by reference numbers 51-58, are illustrated. Physically, as an example, the memory array on a single chip is divided into quadrants, each quadrant including two units that are in part connected together and share a common word line decoding

circuit (y-decode), such as a y-decoder 61 between memory cell units 4 (55) and 5 (56). This memory architecture is similar to that described in aforementioned U.S. patent no. 5,890,192, except there are eight units instead of the four units (quads) illustrated in that patent.

Each of the array units has a bit line decoder (x-decode), such as x-decoder 63 connected to the array unit 5 (56), through which user data is read. Figure 2 is an expanded view of the array unit 5 and its decoders 61 and 63 that respond to addresses on the address bus 47. Connected to the decoder 63 is a circuit 65 that contains sense amplifiers for reading data, a register for storing data being programmed, comparitors used during programming to determine whether addressed cells in the unit 5 have been programmed to the desired state and during reading to determine the states of the cells being read, and control logic to carry out these functions. Two registers 67 and 69 are connected for parallel transfer of user data between them during reading (from 67 to 69) and programming (from 69 to 67). User data is transferred from the data bus 45 and the register 69, one byte at a time, during writing and in the other direction during reading. Each of the other seven array units are similarly connected.

Referring specifically to Figure 2, a portion of an example memory cell array is generally described with respect to the array unit 5. Each row of cells has its own conductive word line (WL) connected through the decoder 61 to corresponding word lines of the adjacent array unit 4. Each of two partial rows 70 and 76 of respective floating gate memory cells 71-75 and 77-81, for example, has its own respective word line 83 and 85. A word line is connected to a gate of each of the cells in a single row, the connected gate being a select gate in a memory cell having a split channel type of structure. Other memory cell structures can be used instead, each having at least one electrically floating gate upon which a level of stored charge is a measure of the state of the cell. A conductive erase line is provided between every other row of memory cells, the line 87 being connected to erase gates of each of the memory cells of each of the rows 70 and 76. Alternate structures do not erase the floating gates to a separate erase gate but rather erase to a region of the substrate such as the cell source diffusions. Bit lines (BL) extend in an orthogonal direction to the word lines, one bit line between each column of array cells, and are connected to the decoder 63. Each bit line is connected to the source and drain diffusions of each of the cells of the columns on either side of the bit line. Detailed examples of suitable memory arrays

are described in the U.S. patents listed in the Background section above but other existing and proposed structures can alternatively be employed in implementations of the present invention.

A block of cells is formed, in the array example being described, from each pair of rows that surround an erase gate, such as the rows 70 and 76 of the array unit 5 (Figure 2) on either side of the erase gate 87, when operating each floating gate in four defined threshold voltage states in order to store two bits of data per floating gate. This is illustrated more generally in Figure 3A, where the block formed of rows 70 and 76 contains a sector's worth of data from the host, plus some overhead information about that data. In a very specific example, the individual blocks are sized to have a capacity of 528 bytes in order to store one 512 byte sector of data, some overhead information about the data and provide some spare bytes, as illustrated in Figure 4. When each floating gate is capable of storing two bits of data, as is the case illustrated in Figure 3A, each block contains 264 floating gates, or 132 in each of the block's two rows. If the memory cell structure has one floating gate per cell formed between source and drain diffusions, then 132 cells are included in each row of each one of the units. But if the cells each have two floating gates, then only 66 cells are necessary in each row.

If, rather than storing two bits of data per floating gate, only one bit is stored per floating gate, twice the number of cells are needed for each operable block. This can be accomplished, in effect, by doubling the length of each row in a manner illustrated in Figure 3B. The rows 70 and 76 are extended to include the respective rows 70' and 76' of the adjacent unit 4 with which unit 5 is paired. The word line of row 70' of the unit 4 is connected through the decoder 61 to the word line of the row 70 of the unit 5, and the word line of the row 76 is connected to the row 76' of unit 5. When a given row's word line is addressed, both components of the word line, in adjacent units 4 and 5, are addressed together. The common erase gate for the rows 70 and 76 is also connected through the decoder 61 to a common erase gate for the rows 70' and 76', in order to cause all of the cells formed of the enlarged block of cells of the rows 70, 70', 76 and 76' to be erased together. Therefore, when operating the memory array floating gates with only two threshold states, the eight units of the array being described are reduced to four operating quadrants that are each formed of adjacent pairs of units (0-1, 2-3, 4-5 and 6-7). This allows the memory array of

Figure 1 to be operated in either of two or four states, according to a command set by the manufacturer, when a proper set of reference threshold levels are also set for program-verify and reading in each unit's circuits 65.

As an alternative to the two state configuration shown in Figure 3B, a single block may be formed of rows that are all within a single unit, such as four rows that are contiguous with each other.

Data is preferably transferred in chunks between the controller buffer memory 35 (Figure 1) and an addressed block of the array memory cells. In a specific example, each chunk contains 66 bytes of data. A chunk of data is programmed to the cells in an addressed block, one time in parallel. When operating in four states, chunks of data 91-94 are stored in one row of a typical block (Figure 3A), and chunks of data 97-100 are stored in the second row of cells of that block. When the memory is operated in two states, individual chunks of data 91'-94' are stored one of the extended rows of cells (Figure 3B), and chunks 97'-100' in the other row of the expanded block of cells that extends across two adjacent units.

Rather than transferring a chunk of user data in parallel between the buffer memory 35 and one of the memory arrays, a data bus within the lines 15 is designed to carry only a few bits data in parallel, one byte in a specific example. This reduces the number of lines 15 that are necessary and, more importantly, reduces the number of user data pads that need to be included on each of the memory array chips in the system. With one byte being transferred at a time, there thus needs to be 66 such transfers for a each chunk. These byte wide user data transfers extend between the buffer memory 35 and, through interface circuits 33 and 39, the memory cell array master register 69, if a block within the array unit 5 is being addressed, or another of the array master registers associated with another unit if a block within that other unit is being addressed.

During programming, bytes of a sector of user data received from the host system are successively transferred into the memory array master register one at a time until a chunk of data has been accumulated, and then the chunk is transferred in parallel to the slave register (such as register 67 for the array unit 5). As the chunk is being transferred out of the master register and into the slave register, bytes of the next chunk of the sector are being transferred into the master register. Chunks are transferred in parallel, one at a time from the slave register, to programming and

verifying circuits (circuit 65 for the array unit 5), which causes the a number of memory cells of an addressed block of the associated array unit to be programmed to their respective target threshold levels. The loading of one chunk into the master register preferably overlaps programming of the previous chunk of data from the slave register by the programming and verifying circuits.

During reading, the process is reversed, memory cell threshold values representative of a chunk of data being read one at a time from one of the array units into its reading circuit (circuit 65 for the array unit 5) where the stored values are converted into data bits. Those successive chunks of data are then transferred one at a time into the slave register for the array unit, thereafter in parallel to the master register, and then one byte at a time over the lines 15 to the data buffer 35 of the controller for transfer to the host system. As a chunk is being transferred out of the master register and to the controller, a new chunk of data is being transferred into the slave register from the read circuits.

Rather than completing the transfer of chunks of data of one sector before commencing another, it is preferred to alternately transfer chunks of multiple sectors between the buffer memory 35 and different ones of the array units 0-7. This is illustrated in Figure 5, where data from four sectors is transferred in a common data stream 101. Memory array units A, B, C and D can be any four of the array units 0-7 of the system of Figure 1. Chunks of data A0+, B0+, C0+ and D0+ are shown to be transferred in the data stream 101 from four respective data sectors 103, 105, 107 and 109 in the buffer memory 35 to the four different memory units A-D. The memory system can, of course, be alternatively designed to transfer fewer or more data sectors together to an equal number of memory cell units. For simplicity in illustration, each sector is shown in Figure 5 to contain only four chunks (chunks A0, A1, A2 and A3 for the sector 103, for example), where the system of Figure 1 has been described above to transfer eight chunks of data for each sector. The principle of operation is the same.

As illustrated in Figure 5, the data stream 101, being transferred in the data bus of the lines 15, is formed from successive bytes of a first chunk A0 of the buffered data sector 103, followed by a first chunk B0 of the sector 105, then a first chunk C0 of the sector 107, and then a first chunk D0 of the sector 109. These chunks are initially stored in the master registers of the respective units to which the data are

being written, transferred in parallel to the slave registers and then written in parallel into respective blocks 111, 113, 115 and 117 of the four memory cell units A-D. All of the chunks A0, B0, C0 and D0 are programmed at the same time into respective memory units A-D, followed by chunks A1, B1, C1 and D1, and so forth. The blocks where chunks of data are written either share a common word line 119 or have the same voltage impressed on the word lines of the addressed blocks. As the initial chunks of data are being transferred through the registers and programmed into the units, subsequent chunks A1, B1, C1 and D1 of the four data sectors are being transferred to the registers of the units. This is followed, in the data stream 101, by another chunk from each of the data sectors 103, 105, 107 and 109, and so forth, until all of the chunks of each of the four buffered data sectors has been programmed into the memory units A-D.

This programming technique has an advantage that multiple chunks of user data may be simultaneously programmed into different units of the memory on a single chip, or, alternatively, into different units spread among two or more memory chips. This is preferred to the simultaneous programming of multiple sectors of data into a common block of one unit since the cells being programmed are, by the technique of Figure 5, physically and electrically separated further from each other. This separation reduces incidents of disturbing the charge on floating gates not intended to be altered. It also spreads out the power supply requirements on the chip. The technique of Figure 5, which operates with memory blocks formed of contiguously positioned memory cells, is also preferred to defining memory blocks in segments across multiple memory units in order to simultaneously program multiple segments of the block without causing disturbs and having increased power requirements.

The technique of Figure 5 has been described with the assumption that all of the multiple data sectors, four in that example, are written in full into the controller buffer 35 before transfer of their data in chunks to the flash memory is commenced. But this transfer can be commenced earlier if all four of the data sectors are being simultaneously written into the buffer memory 35, as can be done with the higher data transfer rates between the controller 11 and the host system. Alternatively, some additional parallelism can be included by loading the four data sectors 103, 105, 107 and 109 into the buffer memory 35 in a time shifted manner, such as by loading data

of the chunk B0 from the sector 105 after the chunk of data A0 has been written and while it is being read out into the data stream 101, and so forth down the line through the data sectors 107 and 109. The data sectors are then loaded into the buffer 35 with a time shift of that necessary to load one chunk of such data. The generation of chunks of data within the stream 101 then needs to wait for only the first chunk of each data sector to be loaded into the buffer 35.

Figure 6A is a block diagram of a circuit that may be included as part of the controller 11 of the Figure 1 memory system in the path of user data being transferred in a stream between the interfaces 25 and 33 during both programming and reading. These circuits participate in carrying out the method described with respect to Figure 5, and also generates and inserts into the data stream an ECC from the data being programmed (ECC generation 37) and make use of the BCPs to avoid writing user data to memory cells within bad columns (BCP processing 36). The micro-controller 23 addresses the buffer memory 109, one byte at a time in a predefined sequence. That sequence is altered by circuits 104 and 106 in response to a signal in a line 108 that indicates a byte address within the flash memory includes a bad column. A multiplexer 121 operates to supply bytes of data from one of sectors A, B, C or D at a time, in response to a control signal in lines 123, as one input to another multiplexer 125. Chunks of data are preferably read from the data sectors stored in the buffer memory 35 in the manner described above with respect to Figure 5. A control signal 123 causes the multiplexer 121 to switch from one data sector to the next, in sequence, each time that a chunk's worth of data bytes has been read from the buffer 35.

The stream of user data bytes at the output of the multiplexer 121 is applied as one input to a multiplexer 125. The multiplexer 125 normally passes that data stream from lines 114 through its output to one input of another multiplexer 112 An exception is when all the user data bytes of a sector have been passed out of the buffer 35, at which time a final series of bytes containing an ECC code for the sector of user data is added onto the end of user data through lines 116. The output of the multiplexer 125 provides the byte wide stream of user data and overhead information of the data for writing into physical blocks of the non-volatile memory.

That stream of user data is normally passed, one byte at a time, through the multiplexer 112 to the flash memory. An exception to this is when a BCP hit signal in

line 108 is active, indicating that the byte of user data is being directed to a location of the flash memory that includes a bad column. In that case, the address applied to the buffer 35 is not incremented by the micro-controller 23 but rather a fill byte contained in a register 110 is inserted into the data stream instead of the usual byte of user data. The next byte of user data from the buffer 35 is delayed until the next transfer cycle when the BCP hit signal is inactive. The bits of the fill byte within the register 110 can be written by the micro-controller 23. The output of the multiplexer 112 provides the byte wide stream of user data and overhead information of the data that is written into physical blocks of the non-volatile memory, as well as possible fill bytes to avoid the effects of bad columns. The memory system being described can operate without the circuit of Figure 6A, or with one of the BCP processing 36 or the ECC generation 37 but not both, but it is preferred to include both functions as shown.

Referring to Figure 6B, additional details of the ECC generation unit 37 of Figure 6A are given. Even though four sectors of data are alternately being transferred in a byte wide data stream from the output of the multiplexer 112 into a single ECC generator circuit 127, the ECC generator 37 is able to generate an ECC separately for each data sector and append the generated code to the end of each sector's data in the last chunk that is sent to the non-volatile memory array for storage. This is done by use of separate registers 133, 134, 135 and 136 for each of the four data sectors being sent to the memory array at one time. A resulting generation is stored in one of these registers by a de-multiplexer 131 or some other effective switching logic. Similarly, the contents of an appropriate one of the registers is connected through a multiplexer 139 as an input to the ECC generation circuit 127. After each byte of data is input to the ECC generation circuit 127 from the buffer 35, that circuit uses an intermediate result of an ECC generation for the same respective data sector A, B, C or D that is stored in the respective one of the registers 133-135, along with the new byte of data, to generate a new intermediate result that is stored back into that same register. This use of the registers is controlled by their input de-multiplexer 131 and output multiplexer 139, which are caused by the control signal in lines 123 to step in sequence between registers in synchronism with the data selecting multiplexer 121. A standard ECC algorithm, such as the Reed-Solomon code, is implemented by the generation circuit 127.

In order to pause the ECC generator 127 when fill bytes are being inserted into the data stream by the BCP processing, the generator 127 is disabled by a combination of the BCP hit signal in the line 108 being active and a signal in the line 116 indicating that the data at the output of the muliplexer 121 is valid. A logic circuit 138 combines the signals 108 and 116 in order to generate a disable signal for the ECC generator 127.

After the ECC generator has received the last byte of data of a sector being stored, the final result is inserted in the last chunk of the data stream by operating the multiplexer 125, in response to a control signal in a line 129, to switch from receiving the input from the multiplexer 121 to receiving the result from the ECC generation circuit 127. The ECC is then stored in the same block of the memory array as the sector of user data from which it was generated. At the end of generating and inserting ECCs into the data stream for each of a group of four sectors of data, the registers 133-136 are reset by a signal in a line 141.

Referring to Figure 6C, additional details of the BCP processing 36 are given. A number of bad column pointers (BCPs) for each of the memory sectors being programmed at a particular time are loaded by the micro-controller 23 into registers 118 from a reserved block of the flash memory (described hereinafter with respect to Figure 11). Four BCPs 0-3 are stored for each of the memory units to which the stream 101 (Figure 5) is being directed. That is, all four BCPs 0-3 for each of these four units have been written by the micro-controller 23 from the reserved block into four planes (sets) of registers 118, each set having four registers. Each of the planes of registers 118 shown in Figure 6C contains BCPs 0-3 for a different unit. A control signal in a line 123 switches between these register planes in order that the BCPs for the unit for which a chunk of data currently in the stream of data from the buffer 35 through the multiplexer 121 is destined. A multiplexer 120 outputs one of the register values to one input of a comparitor 132, whose output is the line 108 that contains the BCP hit signal. One of four registers 122 is also included for each of the units for to which data of the current stream will be written. Each of the registers 122 begins with a count that causes the multiplexer 120 to select BCP0 for its unit. Each time a BCP hit signal in line 108 occurs, circuits 124 and 126 causes the BCP count of the appropriate register 122 to increment by one.

A second input of the comparitor 132 is formed from the contents of registers 128 and 130. These registers contain the current physical memory location for storage of the chunk and byte, respectively, to which the byte of data at the output of the multiplexer 121 is destined for storage within the flash memory. These registers are loaded by the micro-controller 23. Their combined physical memory byte address is then compared by the comparitor 132 with that of one of the BCPs from the registers 118 of the destination memory unit for that data byte. When the first byte of user data for a given unit is present at the output of the multiplexer 121, its physical memory address from registers 128 and 130 is compared with BCP0 of the one of the registers 118 for that unit. If there is a positive comparison, then the BCP hit signal in line 108 becomes active. This results in the current byte of user data of ECC to be held and the fill byte from the register 110 inserted instead. The fill byte will then be written to the byte of memory cells including the bad column, instead of the current data byte.

The BCP hit signal also causes the count in the one of the registers 122 for that unit to switch the multiplexer 120 to select the next BCP1 for that memory unit. The BCP processor is then ready for this process to again take place when the physical memory byte location matches that of the next in order BCP for that unit. A stalled data byte is written during the next cycle, assuming this next physical address comparison does not result in another BCP hit, in which case the data byte is stalled yet another cycle. This process continues until the data stream formed of the current four sectors of data has been transferred to the flash memory, after which the process is repeated for different memory locations stored in the registers 128 and 130 in which the new sectors of data are to be written, and possibly changing the BCPs stored in one or more of the planes of registers 118 if the new data is to be written into one or more different memory units than before.

Referring again to Figure 4, the specification of a block of the memory array that stores a sector of data and the ECC is given. As additional overhead information, a first byte 145 provides basic information of how the remaining data is written. This can include, for example, one or two bits that designate the relative values or polarity with which the stored data needs to be read. This is often changed from block to block and over time, in order to even the wear caused by programming the cells of a block to one of the programmed states from their erased state (which can also be one

of the programmed states). This is described further below with respect to Figures 15-17. A next component is data 147 which is, for what has been described so far, one sector of user data supplied by the host system for storage in the non-volatile memory system. The component 149 is the ECC was generated from the user data 147 during programming, in the manner described above. The number of cells in the individual blocks is chosen to leave a group 151 of cells, capable of storing 8 bytes in this example, as spares. These spares are shown in Figure 4 to be at the end of the block when written or read, which is the case if there are no defects in the rest of the block which require use of some or all of these spares. The number of spare bytes 151 will be less than what is shown in Figure 4 by the number of fill bytes that are inserted into the user data 147 by the BCP processing 36 to avoid bad columns. The insertion of a fill byte into the user data 147 causes subsequent bytes to be delayed by one byte, or moved to the right in the sector data diagram of Figure 4, thus diminishing the number of spare bytes 151 at the end of the data stream by one.

The BCP processing and ECC generation circuits of Figures 6A-C are operated in a reverse manner when sectors of data are being read from the flash memory, except that an entire sector of data is read before data from another sector is read. The ECC generation circuits 37 calculate an ECC from the user data in the sectors as that data are passed one byte at a time through the circuit of Figure 6A from the flash memory to the data buffer 35, in the same manner as described above for writing data. The calculated ECC is then compared with the ECC bytes stored as part of the sector in order to determine whether the sector's user data is valid or not. The BCP processing identifies bytes in the read stream of data that are fill bytes by comparing the physical memory location from which each byte is read with the BCPs for the memory unit is which the sector of data is stored. Those bytes are then eliminated from the read data stream before writing the sector of data into the buffer memory 35.

Referring to Figure 7, the programming of data described in Figures 4-6 is summarized. The dashed line represents the order of chunks of data appearing in the data stream 101 generated by the system of Figures 1-3 for four data sectors A-D, where each sector is transferred in 8 chunks. As can be seen, the first chunks of sectors A, B, C and D appear, in that order, followed by the second chunks of sectors A, B, C and D, and so forth; until the eighth and final chunks of data sectors A, B, C

and D are include. Thereafter, the process is likely repeated with four different sectors of data.

It will be noted that the overhead information that is stored in a block along with a sector of data is limited to information about the data itself and does not include physical overhead information about the block or its operation. Prior memory systems, particularly those which emulate a disk drive by storing 512 byte sectors of user data, have also stored, in the individual user data blocks, information about the block's characteristics in terms of experience cycles, numbers of pulses or voltages required to program or erase the block of cells, defects within the block, and like information about the storage media, in addition to an ECC generated from the data stored in the block. As part of the present invention, however, this type of information about the physical block is stored in another block. As a specific example, individual block overhead records containing such information of a large number of blocks are stored in other memory blocks dedicated to such overhead information and which do not contain user data. Such information of the memory array that affects a number of blocks, such as the identification of bad columns, is stored in yet other memory blocks in order to minimize the memory space that is required for such information. In either case, the overhead information for a given block is read from these other blocks as part of the process of accessing the given block to either read data from it or program data into it. The block overhead information is usually read by the controller prior to accessing the block to read or write user data.

Figure 8 illustrates a few such block overhead data records 151-157 that are stored together in a single memory array block in the format of Figure 4. That is, the multiple overhead records of other blocks form the data 147 of a sector of overhead data that includes the flag byte 145 and ECC bytes 149 generated from the overhead data 147. The block in which the overhead data sector is stored also includes the spare bytes 151 of cells for replacing any cells within a defective column. This sector of overhead data and the block in which it is stored have the same characteristics, and are written and read the same, as described above for user data sectors. The primary difference is the nature of the data 147. In a specific example, each such overhead record contains four bytes of data, resulting in 128 such records being combined together into each defined sector of overhead data.

Figure 9 illustrates example contents of a block overhead record for a good user data block. Byte 0 contains flags including an indication that the user data block is a good one. Byte 1 specifies a voltage for erasing the user data block, and byte 2 a programming voltage. These two voltages are updated over the life of the memory system by the controller in response to the subject user data block requiring an increasing number of erasing or programming pulses to reach the desired respective erase and programmed states. The controller uses this information when erasing or programming, respectively, the subject user data block for which the record 151 contains overhead data. Byte 3 of the overhead data record indicates the extent of use of the subject user data block, either as an experience count that is updated each time the subject user data block is erased and reprogrammed or as the characteristics of one or more tracking cells that are put through the same number of erase and reprogramming cycles as their corresponding user data block. The use of such tracking cells, preferably associated with each of at least the blocks of the memory array designated to store user data, is described in aforementioned U.S. patent application serial no. 08/910,947. Data in byte 3 is periodically rewritten as the number of cycles of use increase or as the characteristics a tracking cell change significantly. The value of byte 3 is used to determine when the associated user data block needs to be retired from service. Use of tracking cells, rather than an experience count, has a significant advantage that the overhead sector data need be rewritten far less frequently, as little as just a few times over the lifetime of the memory array, rather than after each erase/program cycle.

Figure 10 illustrates an overhead record for a user data block that has exceeded its useful lifetime or otherwise has been determined by the controller to be a defective block. The flag byte indicates that the block is defective and a spare block has been assigned to take its place. (Not to be confused with the flag byte 145 (Figure 4) that is at the beginning of the individual data sectors.) The other three bytes of the record specify, as part of the current user data block overhead data, the spare block's address. Thus, the three bytes used for a good block to specify its operating parameters (Figure 9) are efficiently used for this other purpose for a defective block. This minimizes the number of bytes required for the overhead data. When the controller reads this record as part of the process of accessing its user data block, it is quickly determines that the addressed block is defective and the address of the spare

block provided in the record of Figure 10 is then used by the controller to address and access the spare user data block.

This arrangement of blocks thus requires that a number of spare blocks be provided in addition to the number of user data blocks necessary to fill the address space specified for the memory array. The overhead records for these blocks designate their status as spares in the flag byte, and whether they are good or defective spare blocks. If a good spare block, the record contains the same bytes 1-3 as the record of Figure 9. If a defective spare block, the bytes 1-3 need not contain any overhead information of the block since it will never be used.

Any defects of the memory cell array that affect many blocks of cells, such as defective columns as can occur when a bit line has a short circuit to some other element, are stored in other blocks in order to compact the block overhead records. An example shown in Figure 11 is a bad column pointer (BCP) table stored as data in a block having the data format of Figure 4. Such a table is made during the testing phase of the memory system manufacturing process. In this example, up to four BCPs of two bytes each may be stored for each of the memory cell units 0-7. Since eight spare bytes of cells are included as spares 151 in a typical user data block (Figure 4), two bytes of cells may be skipped in a single block in response to the controller reading one BCP from the table of Figure 11. If there are more than four bad columns in a unit, that unit is identified as defective and not used. The memory system may operate so long as at least one unit remains usable. Alternatively, since the column lines of each unit may be segmented, some number of BCPs, such as two, may be stored for each segment or for individual groups of segments. It does, of course, take more memory to store an expanded BCP table.

During operation of the memory, the BCP table of Figure 11 and as many overhead data blocks of Figures 8-10 as space allows, are read from the memory blocks by the controller into its RAM 29 (Figure 1) without deleting that overhead data from the non-volatile memory. This is done upon initialization of the memory system and, if there is not enough room in the RAM 29 for all of the block overhead sectors of data, those that are less frequently accessed by the controller are deleted from the RAM 29 in favor of those that need to be accessed. Since this data can be read much faster by the controller from its own RAM 29 than from a non-volatile memory block, the "caching" of such data in the controller memory speeds up the

process of accessing non-volatile memory blocks containing user data since the overhead data must also be accessed.

The controller 11 may access a number of user data sectors in response to receipt from a host system of a command containing an address, such as a in a cylinder/head/sector format, as in a disk drive, or as a logical block. The controller then calculates a logical address of a beginning block corresponding to the beginning sector address provided by the host. The memory system address space for a given array chip may be expressed as a continuum of logical block addresses (LBAs) that represent all available blocks on the chip in its good memory units for storing user data. The block overhead records are logically arranged in that same order. The controller then first reads the block overhead record in its RAM 29 that corresponds to the first data sector specified by the host while a physical address within the memory array of the user data block is being calculated from its LBA. After accessing the first user data block and its overhead information, the logical block address need not be recalculated for each subsequent block that is addressed for data of the particular file. A simple counter will step through the overhead records that have been organized in the order of their respective user data blocks. If all of the necessary overhead records are not in the controller's RAM 29 at the time of this access, the required overhead data sectors are preferably first read from the non-volatile memory into the RAM 29, including those of substitute blocks whose addresses are contained in the initial overhead records.

One advantage of storing the block overhead data in records separate from the block is the reduced number of times that such records must be rewritten. In the present embodiment, the overhead blocks of data need not be rewritten frequently, perhaps only two or three times during the lifetime of the memory system and some times not at all. Any change to the overhead information record for one block is held as long as possible in a controller memory before rewriting the overhead data sector in which record exists, and then it can be done in the background without being part of an erase or programming cycle. But when the overhead data of a block is stored in that block, the overhead data must be reprogrammed each time the block is erased. In the examples described herein where only one sector of user data is stored in a block, overhead data would have to be rewritten in one block every time a sector of user data is written into that memory block. This can also require the overhead information to

be written twice into the same block, once before writing the user data and once after doing so, in order to compensate for effects due to programming adjacent cells, particularly when being programmed in multiple states where the tolerance of such effects is less.

Although the memory system examples being described store only one sector of user data in each of the individual blocks, various aspects of the present invention apply equally well where two or more sectors of data, each with its flag and ECC bytes, are stored in individual blocks of the memory cell array.

Figure 12 provides an example utilization of the individual blocks of a memory array chip having eight units of blocks. The same boot information is stored in a specified block of each of the units, usually the first block. Upon initialization of the system, the controller firmware causes the first block of unit 0 to be read but if that block is not readable or its stored data corrupted, the first block of unit 1 is accessed for reading, and so on, until valid memory system boot information is read by the controller. It is then stored in the controller RAM 29. There are also a number of reserved blocks, such as reserved blocks 0-7 being shown in Figure 12, in which data desirable for operation of the memory array chip are stored. A copy of the data of each of the reserved blocks is provided in a different memory unit than the primary copy as insurance against a defective unit. The data format of each of the reserved blocks is that illustrated in Figure 4.

Part of the boot information is a physical address of reserved block 0 and its copy, which, in a specific example, contains the bad column pointers of Figure 11, an identity of any unusable unit(s), and physical mapping characteristics including which of the blocks of good units are reserved, those designated for user data, those designated for overhead data ("O.H. Data") according to Figures 8-10, and those designated as spare blocks. Reserved block 0 also contains read and write control parameters particular to its memory array chip.

The other reserved sectors contain information that is useful to the controller to operate the memory array chip on which they are located, or useful to the entire memory system. Reserved block 1, for example, can contain data of system parameters that appears on only the first logical memory array chip of the system, including a specification of the data interface 13 of the controller with the host system. Another reserved block may contain manufacturing information, for

example. The data stored in the boot information and reserved blocks are written as part of the manufacturing process. This data may be made modifiable after manufacture, either in part or in total.

Figure 13 illustrates a desired step in the manufacturing configuration process. Data about each memory array chip is read by the controller 11 from the reserved sector 0 of each chip and assembled into a common file illustrated in Figure 14 that is then written back into reserved sector 2 of the first logical chip 17 of the memory system. This process can be provided by firmware included in the program memory 27 (Figure 1) of the controller. Upon invoking this firmware routine during the system configuration, data of the number of good user data blocks is read from each of the system's memory chips. That information then becomes a part of the system record of Figure 14. Logical block address (LBA) ranges are also added to that record, the first user block of the first logical memory chip 17 being assigned an address of 0000. The ending LBA is also recorded as part of the record, being the beginning LBA of zero plus the number of user data blocks on the first chip. The logical memory space for the next memory chip 19 is noted by its number of good user data blocks and its ending LBA, which is one more than the ending LBA of the first memory chip plus the number of good user data blocks in the second chip. This process continues until the table of Figure 14 is completed for all the memory array chips.

In a specific example, the entry in the table of Figure 14 for each memory chip in the system includes a one byte physical chip number and three bytes for the ending LBA. This results in the merged system LBA table being very short and quick to access. When accessing a particular user data block of the system in response to an address from a host system, as a particular example, the controller 11 first calculates a corresponding logical block address. That LBA is then compared with the table (Figure 14) in reserved block 2 of the first logical memory chip 17 to determine on which of the chips the addressed block lies. The LBA is of the immediately preceding logical chip is then subtracted from the calculated LBA. The physical block address on that chip is then calculated by the controller reading the designated chip's reserved sector 1 information to shift this differential LBA into a corresponding block of the designated chip that has been designated for user data storage.

In addition to storing a record of the number of good units and sectors from which the merged table of Figure 14 is formed, each of the memory array chips includes other information about its respective array that the controller uses to operate it. This other information can also be stored in the reserved sector 0 of each chip array. Another feature of the present invention is that multiple array chips that are used together with a common controller in one system need not each have these same characteristics. The controller 11 preferably reads the characteristics stored in reserve sector 0 of an individual chip into its RAM 29 before accessing that chip. This can be once when the memory system in initialized, provided the controller RAM 29 is large enough, and then accessed from RAM during operation of the memory. Alternately, this information can read from a particular chip just before it is accessed each time and then stored in the RAM 29 only during the time of such access. It is a significant advantage to be able to form a memory system with two or more memory chips that have different operating characteristics. This greatly simplifies the manufacturing process since memory chips from different process batches, different foundries, different manufacturers, and so on, can be combined to operate well together. Testing of the chips in order to batch together those having the same values of these characteristics is no longer necessary.

These characteristics that can be different among memory chips include various optimal voltages and timing, as well as minimum and maximum permitted values, for use to program, read, erase, scrub and refresh the chip. The optimal number of programming pulses, their durations, frequency and magnitudes, as well as operating characteristics of voltage pumps on the chip, including ranges of maximum and minimum values can also be included. The number of sectors per unit and other information needed by the controller to translate a logical block address to a physical one within the particular chip are also stored. Pointers to spare units and blocks within units may also be included. Other information includes pointers to the physical locations of the reserved sectors so that they need not all be stored in the same blocks in each memory chip of a system. The number of user data blocks for which data is included in each of the overhead blocks (Figure 8) can also be stored. Inclusion of the bad column pointers (BCPs) in reserved sector 0 has already been described. A duplicate set of certain operating parameters can be also be included for different chip supply voltages, such as 3 and 5 volts. The controller can then tailor its access to

match the characteristics of each chip in the system, which need not, as a result, be made the same.

The flag byte 145 (Figure 4) of a sector of data is described with respect to Figure 15. In a specific example, the most significant bits 0 and 1 of that byte give a factor by which all the following bits in that sector are transformed. These two bits are preferably randomly selected during the writing process for each sector of data by the micro-controller 23 or fixed logic such as a state machine, prior user data of the sector being read into the data stream 101 (Figure 5). Alternatively, the transformation bits may be assigned by sequencing through all the possible combinations in order. These transformation bits are positioned at the beginning of the remaining flag bits, which are in turn inserted into the stream of data prior to the user data and ECC bytes that form the sector being stored. The purpose of so transforming the data stored in the flash memory is to cause the memory cells to be programmed to different states for the same data. This prevents uneven wear of the various blocks of the memory that can occur over the life of the memory, such as when substantially the same data file is repetitively written to the same group of blocks. When the data sector is read, these transform bits are first read and then used to transform the raw data read from the memory back to the data that was originally received by the memory for storage. Two transform bits are used for a four state memory operation, more being used when the number of storage states of the individual memory cells is made higher. If the memory is operating in a two state mode, however, only one transformation bit is used.

Figure 16 illustrates an example process of transforming received data before being stored in the flash memory. This process can be carried out primarily by the micro-controller 23 under the control of its firmware, or can be executed by the use of adder circuits and related digital elements. Each byte of data from the buffer 35 has its bits separated into four pairs, and each bit pair is combined in respective adders 175-178 with the two bit transform (for a four state flash memory) that is held at 181. The transformed bit pairs are then recombined into a single byte that is applied to one input of a multiplexer 183. These transform bits are also used to transform, in adders 185-187, six fill bits indicated at 189. The transform bits themselves are combined with the transformed fill bit pairs to form the first byte 145 (Figure 4) of a data sector, that is applied to a second input of the multiplexer 183. A control signal in a line 191

then switches the multiplexer 183 to output in circuits 193 successive bytes of data wherein the first byte of the sector is in the form of Figure 15 received through the multiplexer 1 input and subsequent bytes of transformed data are received through the 0 input until the entire sector of data is transformed and sent to the flash memory. It is this transformed data, rather that the raw data received from the host, that is subjected to the BCP processing and ECC generation of Figure 6A.

An inverse transform takes place when the sector of data is read from the flash memory, as illustrated in Figure 17. Four subtraction elements 195-198 receive respective pairs of bits of a data byte received in the path 194 form the flash memory. The two transform bits for the particular data sector are stored at 199, which may be a register. The transform bits, which are the first two bits of the first byte received for the sector, are stored at 199 in response to receiving the first byte control signal in the line 191. These two bits are combined with each of the received byte pairs in the subtracting elements 195-198 in a manner to transform them back to the raw received data. The inverse transform of the sector data after the first two bits commences immediately after those transform bits are read. There is certainly no need to read the transform bits in a separate operation. The remaining 6 bits of the first byte are immediately transformed by its three pairs passing through three of the subtracting elements 195-198 and then being stored at 201 in response to the control signal in the line 191. As subsequent bytes of the data sector are received, one at a time, the contents at 199 and 201 remain unchanged while the inverse transform of the data bytes is made and the result output to the buffer memory 35 over a circuit 203. This output data is then the same as that which was initially received by the memory system and stored in a transformed form.

The least significant bits 4-7 of the flag byte 145 illustrated in Figure 15 are used by the controller to determine whether a block is erased or not. In one example of a memory system, the erased state is also one of the programmed states, so some care needs to be taken to distinguish between them. When necessary to determine whether a block in this type of system is erased from the data in the block, therefore, the sampling of one bit of data, for example, cannot itself provide the answer. Therefore, in one example, fill bits 189 (Figure 16) that are written into the first byte of a data sector are either a fixed known pattern or made to have a pattern that assures those bits are not all equal to the value of the erased state. The controller reads any

data sector in the block and looks for whether the pre-erase bits 173 stored in the corresponding cells of the block either are the fixed values of the fill bits 189 or, if there is no known pattern, at least one of them (and preferably more) is in a programmed state that is not the erased state. As a further check, it may be determined whether an ECC code is present in the data sector. If either or both of these conditions are determined to exist, it is concluded that the block is not erased. But if the bit values of data read from the flag byte bit locations 4-7 and/or the ECC all have the same value equal to that of the erased state, this indicates that the block is in its erased state.

Although specific examples of various aspects of the present invention have been described, it is understood that the present invention is entitled to protection within the scope of the appended claims.

IT IS CLAIMED:

1. A method of operating a re-programmable non-volatile memory system having its memory cells organized into distinct blocks of simultaneously erasable cells, comprising:

identifying a first group of said blocks for storing user data and a second group of said blocks for storing information of the characteristics of said first group of blocks,

storing, in individual ones of the first group of said blocks, user data without data of the characteristics of said first group of blocks, and

storing, in individual ones of the second group of said blocks, a plurality of records of characteristics of individual ones of the first group of blocks but without storing user data in the second group of blocks.

2. The method of claim 1, wherein storing the plurality of records in individual ones of the second group of said blocks includes storing a redundancy code generated from records written therein.

3. The method of claim 1, wherein storing, in the second group of blocks, the plurality of records of characteristics of the first group of blocks individually includes storing programming and reading characteristics of a corresponding one of the first group of blocks.

4. The method of claim 3, additionally comprising:

reading the records from a plurality of said second blocks and storing the read records in a controller memory, and

when accessing one or more of the first group of blocks to program user data therein or to read user data therefrom, reading from the controller memory those of the records stored therein which contain the characteristics of said one or more of the first group of blocks being accessed.

5. The method of claim 4, wherein records of at least one of the plurality of second blocks stored in the controller memory which have the longest time since

being read are removed therefrom when a limited capacity of the controller memory requires space to be made for records from another of the plurality of said second blocks to be stored therein in order to be read when one or more of corresponding ones of the first group of blocks is being accessed.

6. The method of claim 4, wherein, when a plurality of the first group of blocks with successive addresses are being accessed, an address of a record stored in the controller memory that corresponds to a first of the addressed block within the first group of blocks is calculated and remaining records within the controller memory that correspond to others of the plurality of the first group of blocks being accessed are addressed by incrementing from one record address to another.

7. The method of claim 1, wherein storing, in the second group of blocks, the plurality of records of characteristics of the first group of blocks individually includes storing an indication of whether a corresponding block within said first group is defective or not, and, if so, storing in the second group of blocks an address of a substitute block, and, if not, storing in the second group of blocks operating characteristics of the corresponding block within said first group.

8. The method of claim 7, wherein storing operating characteristics includes storing any of programming, reading, erase or wear characteristics of the corresponding block within the first group.

9. The method of claim 1, wherein storing, in the second group of blocks, the plurality of records of characteristics of the first group of blocks individually includes storing indications of locations of any bad columns that extend through corresponding blocks within said first group.

10. The method of claim 9, wherein storing user data into individual ones of the first group of blocks and storing block characteristic records into individual ones of the second group of blocks includes skipping any bad column locations in the respective blocks.

11. The method of any one of claims 1, 3 or 7, wherein characteristics of the user data being stored in individual ones of the first group of said blocks are additionally stored therein along with the user data to which such characteristics relate.

12. The method of any one of claims 1, 3 or 7, wherein the method is practiced when the memory system is enclosed within a card that is removably connectable to a host system.

13. The method of any one of claims 1, 3 or 7, wherein the memory cells within at least a plurality of said blocks are individually operated with more than two storage states in order to store more than one bit of data per memory cell.

14. The method of any one of claims 1, 3 or 7, wherein the memory cells within at least a plurality of said blocks are individually operated with exactly two storage states in order to store exactly one bit of data per memory cell.

15. A method of operating a re-programmable non-volatile memory system having its memory cells organized into distinct blocks of simultaneously erasable cells, comprising:

designating a first group of said blocks for storing user data and a second group of said blocks for storing information of the characteristics of said first group of blocks,

storing, in individual ones of the first group of said blocks, user data plus characteristics of the user data being written therein but not including characteristics of said first group of blocks, and

storing, in individual ones of the second group of said blocks, a plurality of records of characteristics of individual ones of the first group of blocks but without storing either user data or characteristics of the user data into the second group of blocks.

16. The method of claim 15, wherein storing the user data characteristics in individual ones of the first group of blocks includes storing redundancy codes generated from the user data stored therein.

17. The method of claim 15, wherein storing the plurality of records in individual ones of the second group of said blocks includes storing a redundancy code generated from records written therein.

18. The method of claim 15, wherein storing user data in individual ones of the first group of said blocks includes simultaneously writing user data into a plurality of the first group of blocks until at least one user data sector is written into each of the plurality of blocks in the first group, the individual data sectors including at least one characteristic of its user data.

19. The method of claim 18, wherein said at least one characteristic of the user data that is included as part of sectors of data includes redundancy codes that have been generated from user data while the user data is being transferred in a stream to said individual blocks within said first group, individual ones of the redundancy codes being appended to ends of the user data from which they are generated to form sectors of data.

20. The method of claim 19, wherein the redundancy codes are generated in a single circuit used for all of the plurality of the first group of blocks to which user data are simultaneously being transferred.

21. The method of claim 18, wherein said at least one characteristic includes one or more bits by which the user data of within a data sector is transformed before being stored.

22. The method of claim 18, wherein said at least one characteristic includes a plurality of bits of varying values independent of the user data and which are used to determine whether a sector of data including said plurality of bits is stored in respective blocks within said first group.

23. The method of any one of claims 15 or 18, wherein the memory cells within at least a plurality of said blocks are individually operated with more than two storage states in order to store more than one bit of data per memory cell.

24. The method of any one of claims 15 or 18, wherein the memory cells within at least a plurality of said blocks are individually operated with exactly two storage states in order to store exactly one bit of data per memory cell.

25. A method of managing a non-volatile flash memory system having its memory cells organized into distinct blocks of simultaneously erasable cells, comprising:

storing within individual ones of said blocks at least one sector of data including user data, a redundancy code generated from the user data and a plurality of bits at a beginning of the sector that define a function by which the user data and redundancy code are transformed prior to being stored, and

storing at least one characteristic of said individual ones of said blocks in at least one block other than said individual ones of said blocks, said at least one other block storing said at least one characteristic of a plurality of said blocks storing user data.

26. The method of claim 25, wherein said at least one characteristic includes an indication of whether a corresponding one of said individual ones of said blocks is defective or not, and, if so, an address of a substitute block, and, if not, programming characteristics of the corresponding block.

27. A non-volatile memory system, comprising:

an array of floating gate memory cells formed in blocks of cells that are simultaneously erasable together, a first plurality of the blocks being designated to store user data and a second plurality of the blocks being designated to individually store records of characteristics of first plurality of blocks,

a controller memory separate from the array of floating gate memory cells in which at least some of the records from the second portion of the blocks are

temporarily stored, said controller memory being characterized by having a faster access time than that of the floating gate memory cell array, and

a controller adapted to communicate sectors of user data between a host and the first plurality of the memory cell blocks while utilizing records in the controller memory from the second plurality of blocks that correspond to those of the first plurality of blocks with which user data are communicated.

28. The memory system of claim 27, wherein the floating gate memory cell array and the controller are enclosed in a card having electrical contacts thereon that match electrical contacts of a socket of a host system, the card thereby being removably connectable with the host.

29. The memory system of claim 27, wherein the floating gate memory cell array and the controller are embedded within a package containing the host system.

30. A non-volatile memory system, comprising:

at least two floating gate memory cell arrays formed on at least two respective integrated circuit chips, wherein the memory cells of each of the memory cell arrays are grouped into a number of blocks designated to individually store a given quantity of user data, and further wherein the number of such available blocks is different in individual ones of said at least two memory cell arrays,

a memory controller, and

a record stored in the memory system which contains non-overlapping logical address assignments of the blocks of each of the memory cell arrays, thereby to allow the controller to determine from a logical block address which of the memory arrays a corresponding physical block lies.

31. The memory system of claim 30, wherein the logical address assignment record is stored within one of said at least two memory chips.

32. The memory system of claim 30, wherein said at least two integrated circuit chips and the controller are positioned within an enclosed memory card having

electrical contacts thereon that match electrical contacts of a socket of a host system, the card thereby being removably connectable with the host.

33. The memory system of claim 30, wherein said at least two integrated circuit chips and the controller are embedded within a host system.

34. The memory system of claim 30, wherein the controller is formed on one of said at least two integrated circuit chips.

35. The memory system of claim 30, wherein the controller is formed on an integrated circuit chip without a floating gate memory cell array and that is in addition to said at least two integrated circuit chips.

36. A method of manufacturing a non-volatile memory system, comprising:

installing and interconnecting at least first and second integrated circuit chips that individually include an array of non-volatile floating gate memory cells, wherein said at least first and second circuit chips individually contains stored therein a record of at least a number of blocks of capacity of its memory cell array for storing user data, and

merging the memory array capacity records of each of said at least first and second circuit chips to form a merged record on said first circuit chip of contiguous ranges of logical memory block addresses assigned to the memory cell arrays of each of the at least first and second memory array chips.

37. The method of claim 36, wherein the number of blocks of memory capacity for storing user data is different among said at least first and second circuit chips.

38. The method of claim 36, wherein the number of blocks of memory capacity for storing user data is the same among said at least first and second circuit chips.

39. The method of claim 36, additionally comprising installing said at least first and second circuit chips within an enclosed memory card having electrical contacts thereon for engaging contacts of a host connector.

40. The method of claim 36, additionally comprising embedding said at least first and second circuit chips within a package containing the host system.

41. A method of operating a flash EEPROM system having its memory cells organized into distinct blocks of a number of simultaneously erasable cells capable of storing a given quantity of data, comprising:

providing the memory system with a controller and a plurality of physically distinct arrays of said memory cells that are individually organized into said memory blocks,

storing in one of the memory blocks a record of a number of blocks available in each of said plurality of memory cell arrays for storing user data and non-overlapping ranges of contiguous logical addresses assigned to said number of user data blocks of the individual memory cell arrays, and

locating a physical address of a memory cell block at least in part by accessing the record with a logical memory cell block address in order to determine one of the plurality of memory cell arrays in which the addressed memory cell block resides.

42. The method of claim 41, wherein the number of blocks of memory capacity for storing user data is different among at least two of the plurality of memory cell arrays.

43. The method of claim 41, wherein the number of blocks of memory capacity for storing user data in each of the plurality of memory cell arrays is the same.

44. The method of claim 41, wherein the plurality of memory arrays are enclosed in a memory card having electrical contacts thereon for engaging contacts of a host connector.

45. The method of claim 41, wherein the plurality of memory arrays are embedded within a package containing a host system.

46. The method of claim 41, wherein the memory cells within at least some of said memory cell blocks are individually operated with more than two storage states in order to store more than one bit of data per memory cell.

47. The method of claim 41, wherein the memory cells within at least some of said memory cell blocks are individually operated with exactly two storage states in order to store exactly one bit of data per memory cell.

48. A method of operating a re-programmable non-volatile memory system having floating gate memory cells organized into distinct blocks of a number of simultaneously erasable cells capable of storing a given quantity of data, the blocks of cells being further organized into a plurality of units, comprising:

receiving and temporarily storing at least a given number of sectors of user data to be programmed into the memory system,

simultaneously programming a chunk of user data from each of the given number of temporarily stored sectors of user data into different blocks of memory cells within a number of different memory cell units equal to said given number, each chunk being a fraction of a sector of user data equal to one-half or less, and

repeating the simultaneous programming of chunks of user data until all the data of each of the given number of temporarily stored sectors has been programmed into the different blocks within the given number of memory cell units.

49. The method of claim 48, additionally comprising, prior to programming chunks of data into blocks of memory cells, alternately transferring one chunk at a time in sequence from the sectors of temporarily stored user data into a plurality of storage registers equal to said given number, and thereafter the programming includes transferring the chunks of data stored in the storage registers in parallel into the blocks of memory cells within the given number of units.

50. The method of claim 49, wherein each of the given number of sectors of user data is received and temporarily stored before commencing to transfer chunks thereof into the storage registers.

51. The method of claim 49, wherein only a portion of each of the given number of sectors of user data is received and temporarily stored before commencing to transfer chunks thereof into the storage registers.

52. The method of claim 49, additionally comprising generating a redundancy code for each of the given number of sectors of data as the individual chunks of data are transferred from temporary storage into the storage registers, including using a common generating circuit for each of the given number of sectors of user data, separately storing intermediate results of the redundancy code generations in a separate code register for each of the given number of sectors of data and combining the stored intermediate results of one sector of data with a new quantity of data for the same sector.

53. The method of claim 52, additionally comprising including the redundancy code generated for each of the sectors of data in a final chunk of user data that is transferred to the storage registers.

54. The method of claim 49, additionally comprising maintaining a table of defective column addresses for each of the plurality of memory units, repetitively comparing destination addresses of the chunks of data with the column addresses in said table, and, in response to a positive comparison, inserting bits into the chunks prior to programming them into the memory blocks in a manner that the inserted bits are programmed into memory cells in defective columns.

55. The method of claim 49, wherein the memory cells within at least a plurality of said blocks are individually operated with more than two storage states in order to store more than one bit of data per memory cell.

56. The method of claim 49, wherein the memory cells within at least a plurality of said blocks are individually operated with exactly two storage states in order to store exactly one bit of data per memory cell.

57. A method of operating a re-programmable non-volatile memory system having floating gate memory cells organized into distinct blocks of a number of simultaneously erasable cells capable of storing a given quantity of data, the blocks of cells being further organized into a plurality of units, comprising:

receiving and temporarily storing in a buffer memory at least a given number of sectors of user data to be programmed into the memory system,

moving data in a stream from one of the given number of sectors of user data in the buffer at a time to a respective one of a given number of storage registers at a time, and

thereafter moving the user data from the given number of storage registers in parallel to respective ones of a given number of memory cell blocks that are located within different ones of a given number of said units.

58. The method according to claim 57, wherein moving data from the storage registers to the memory cell blocks includes moving one chunk of a sector of user data from each of the given number of registers, wherein the amount of data in a chunk is equal to one half or less of the amount of data in a sector.

59. The method according to claim 58, wherein moving data from the buffer memory to the storage registers includes moving one chunk at a time alternatively from the given number of sectors of user data stored in the buffer memory.

60. The method according to claim 57, wherein moving data from the storage registers to the memory cell blocks includes moving a full user data sector from each of the given number of registers.

61. The method according to claim 60, wherein moving data from the buffer memory to the storage registers includes moving data from one sector at a time

in sequence from the given number of sectors of user data stored in the buffer memory.

62. The method according to claim 57, which additionally comprises, prior to commencing moving the stream of data, generating a data transformation bit field for each sector of user data and using that bit field to transform the user data that is moved in a stream, and further comprising inserting the generated transformation bit field into each of the given number of sectors of user data at its beginning.

63. The method according to claim 57, wherein moving data in a stream includes generating a redundancy code from the stream of user data of the individual sectors and appending the generated code to the ends of the user data from which they are generated.

64. The method according to claim 59, wherein moving data in a stream includes generating a redundancy code from the stream of user data of the individual sectors and appending the generated code to the ends of the user data from which they are generated, the redundancy code generation including separately storing intermediate results of the redundancy code generations in a separate code register for each of the given number of sectors of data and combining the stored intermediate results of one sector of data with a new quantity of data for the same sector.

65. The method according to claim 57, wherein moving data in a stream includes inserting bits into the data stream for storage within cells in any defective columns of the memory blocks, the inserted bits shifting the user data thereafter.

66. The method according to claim 59, wherein moving data in a stream includes inserting bits into the data stream for storage within cells in any defective columns of the memory blocks, the inserted bits shifting the user data thereafter, the bit insertion including referencing addresses of any defective columns within each of the given number of memory cell blocks into which user data is moved from the storage registers.

67. The method of any one of claims 57-62, wherein the memory cells within at least a plurality of said blocks are individually operated with more than two storage states in order to store more than one bit of data per memory cell.

68. The method of any one of claims 57-62, wherein the memory cells within at least a plurality of said blocks are individually operated with exactly two storage states in order to store exactly one bit of data per memory cell.

69. The method of any one of claims 57-62, wherein the memory cells within a first group of a plurality of said blocks are individually operated with more than two storage states in order to store more than one bit of data per memory cell, and wherein the memory cells within a second group of a plurality of said blocks different from said first group are individually operated with exactly two storage states in order to store exactly one bit of data per memory cell.

70. The method of any one of claims 57-62, wherein the sectors of user data stored in the memory cell blocks do not include characteristics of the memory cell blocks in which they are stored.

71. A non-volatile memory system, comprising:

an array of floating gate memory cells formed into blocks of cells that are simultaneously erasable together,

a plurality of data registers,

a first data transfer circuit that moves data in parallel between the plurality of data registers and respective distinct blocks of the memory cell array,

a buffer memory capable of storing a plurality of sectors of user data at the same time,

a second data transfer circuit that moves user data in a stream between the buffer memory and one of the data registers at a time,

a redundancy code circuit positioned in the path of the data stream to generate a redundancy code in real time from the data stream, and

a defective column circuit positioned in the path of the data stream to adjust the length of the stream to avoid defective columns within the memory cell array.

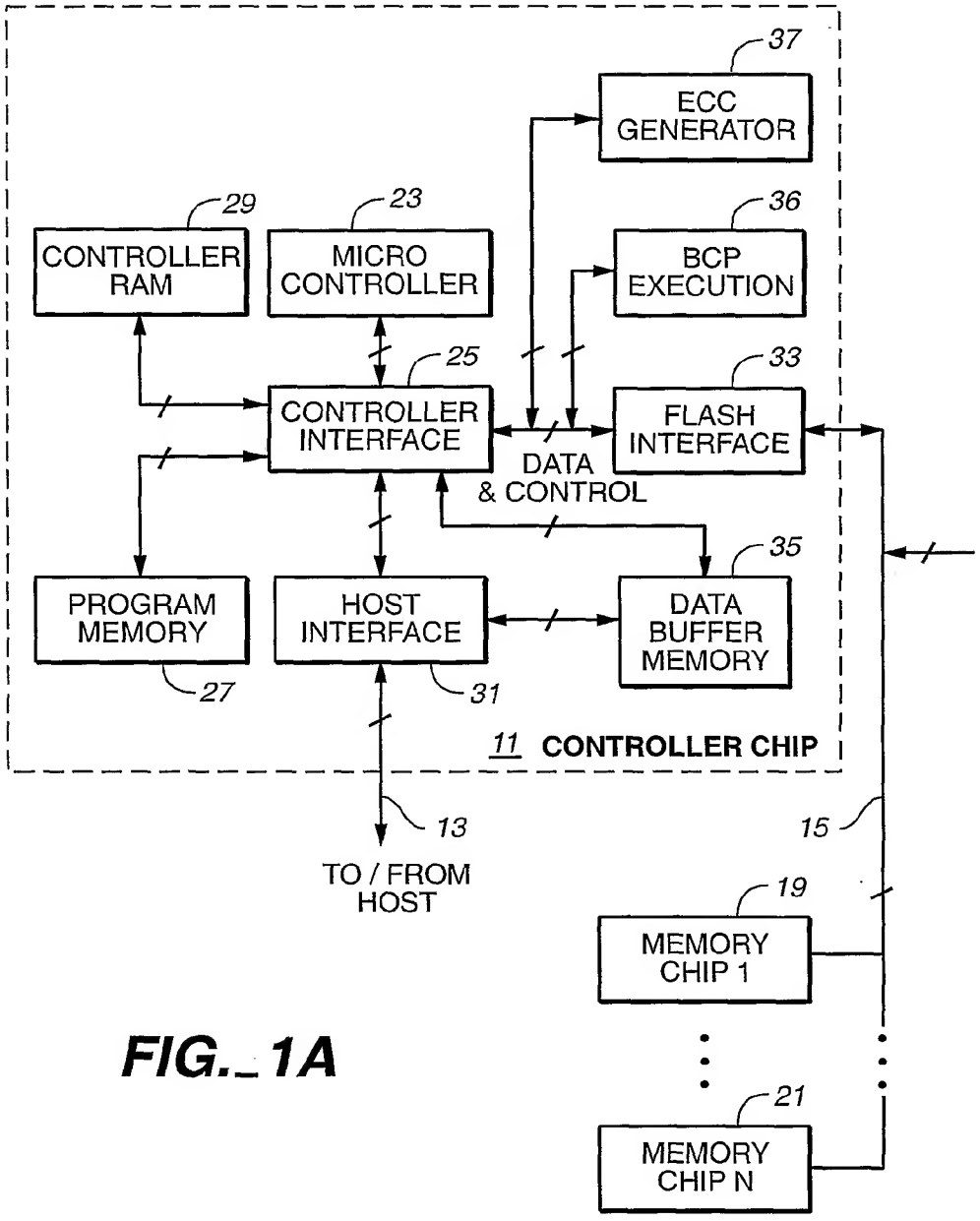

37
ECC GENERATOR
29
CONTROLLER RAM
23
MICRO CONTROLLER
36
BCP EXECUTION
25
CONTROLLER INTERFACE
33
FLASH INTERFACE
DATA & CONTROL
35
PROGRAM MEMORY
HOST INTERFACE
DATA BUFFER MEMORY
27
31
11 CONTROLLER CHIP
13
TO / FROM HOST
15
19
MEMORY CHIP 1
21
MEMORY CHIP N

FIG._1A

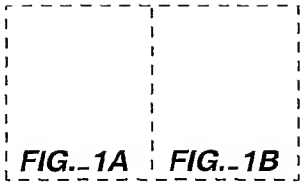


FIG._1A
FIG._1B

FIG._1

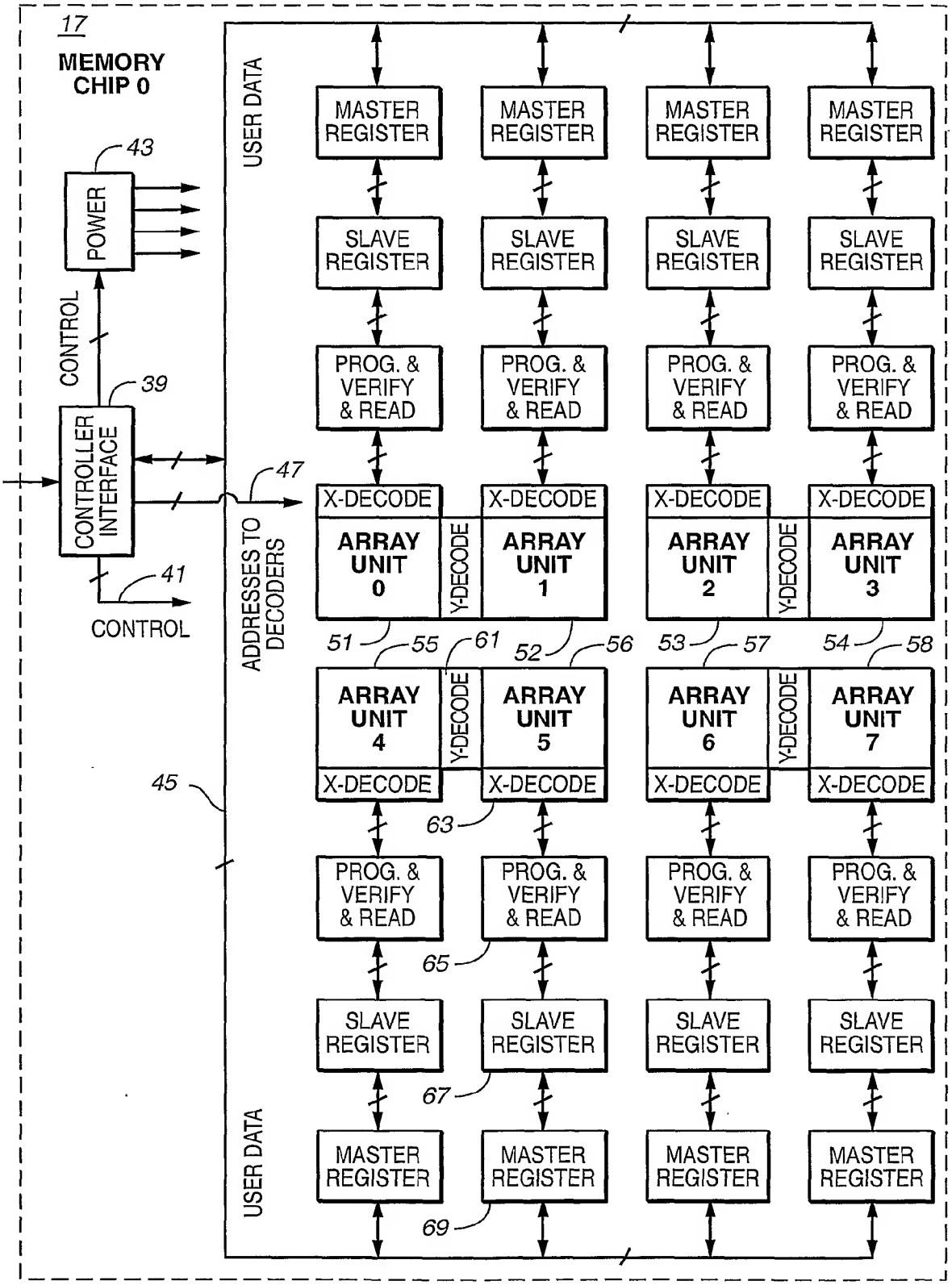

17
MEMORY CHIP 0
43
POWER
CONTROL
39
CONTROLLER INTERFACE
41
CONTROL
47
ADDRESSES TO DECODERS
45
USER DATA
MASTER REGISTER
SLAVE REGISTER
PROG. & VERIFY & READ
X-DECODE
ARRAY UNIT 0
ARRAY UNIT 1
ARRAY UNIT 2
ARRAY UNIT 3
Y-DECODE
51
55
61
52
56
53
57
54
58
ARRAY UNIT 4
ARRAY UNIT 5
ARRAY UNIT 6
ARRAY UNIT 7
63
65
67
69

FIG._1B

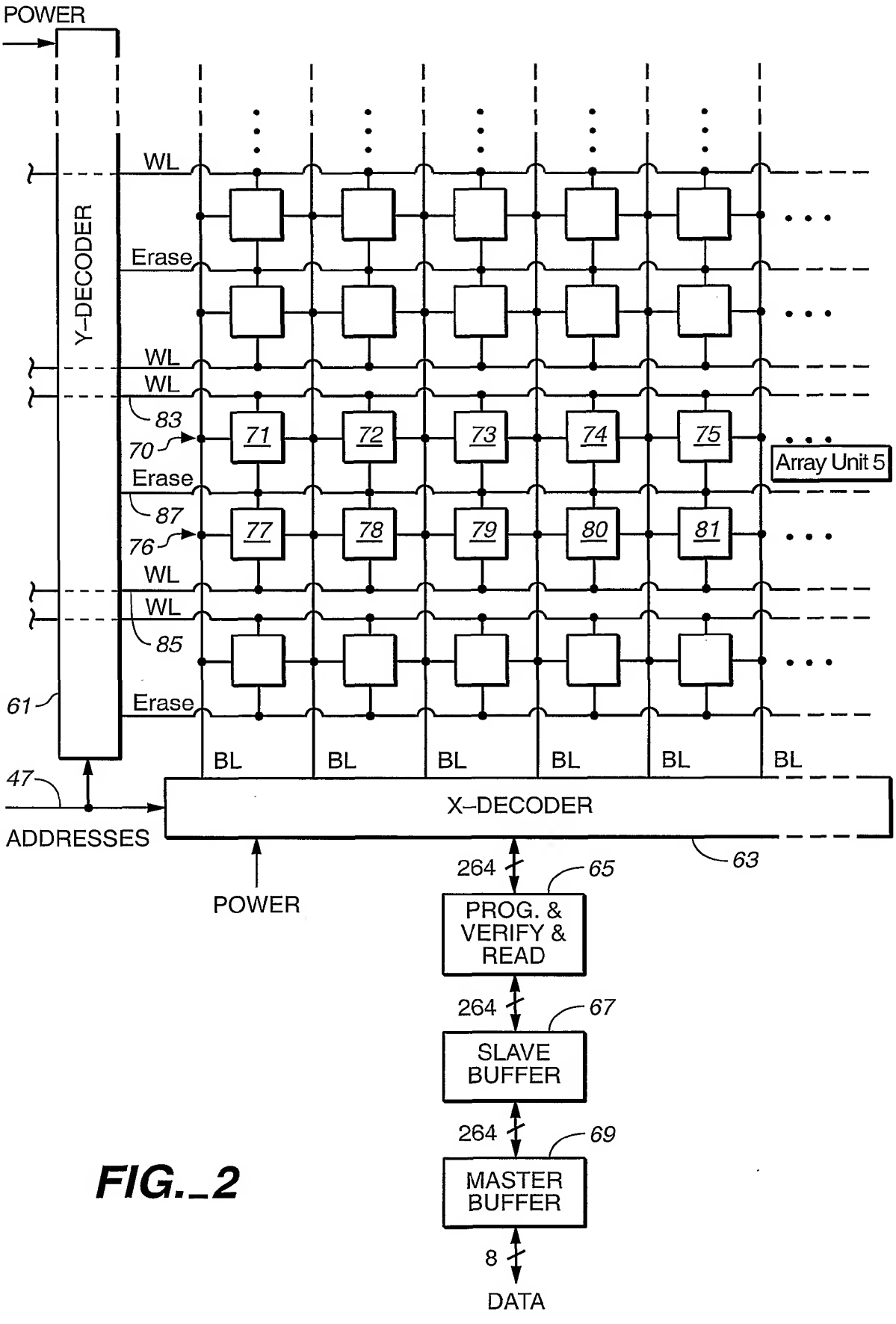

POWER
Y-DECODER
WL
Erase
WL
WL
83
70
71
72
73
74
75
Array Unit 5
Erase
87
76
77
78
79
80
81
WL
WL
85
61
Erase
BL
BL
BL
BL
BL
BL
47
X-DECODER
ADDRESSES
POWER
63
264
65
PROG. & VERIFY & READ
264
67
SLAVE BUFFER
264
69
MASTER BUFFER
8
DATA

FIG._2

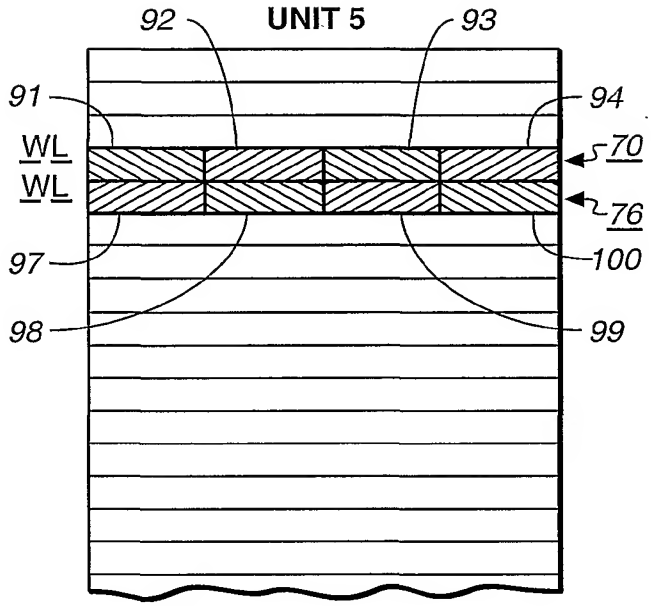


FIG._3A

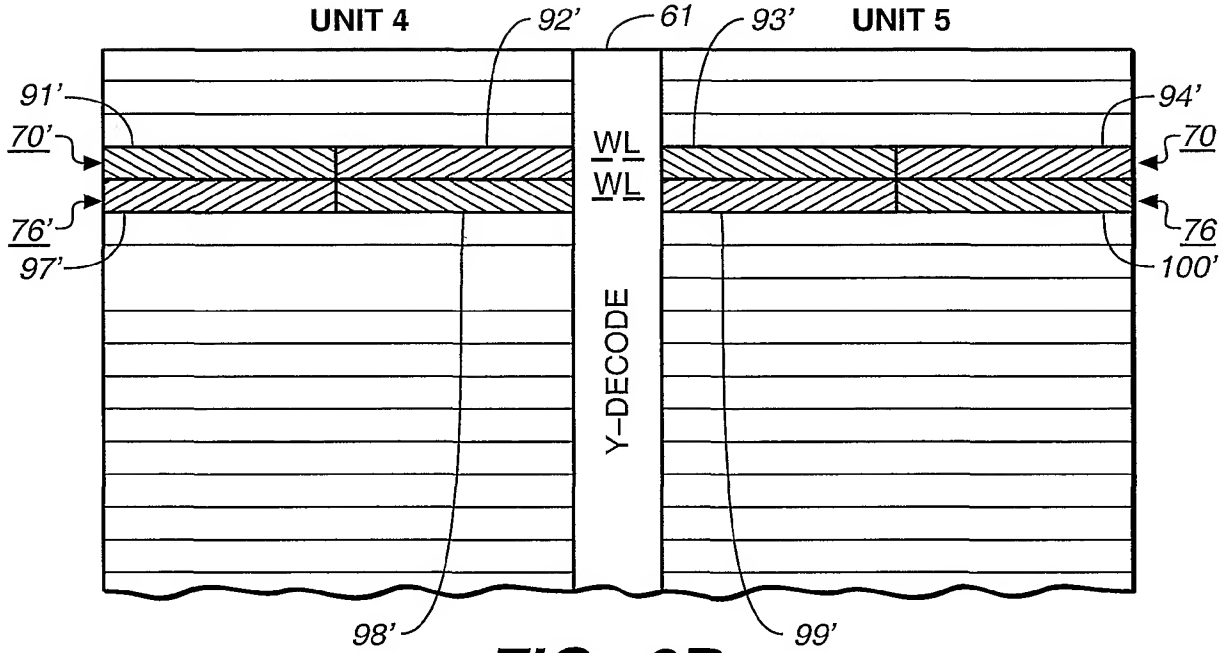


FIG._3B

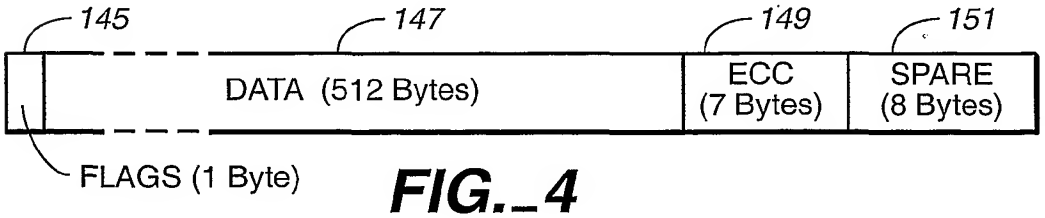


FIG._4

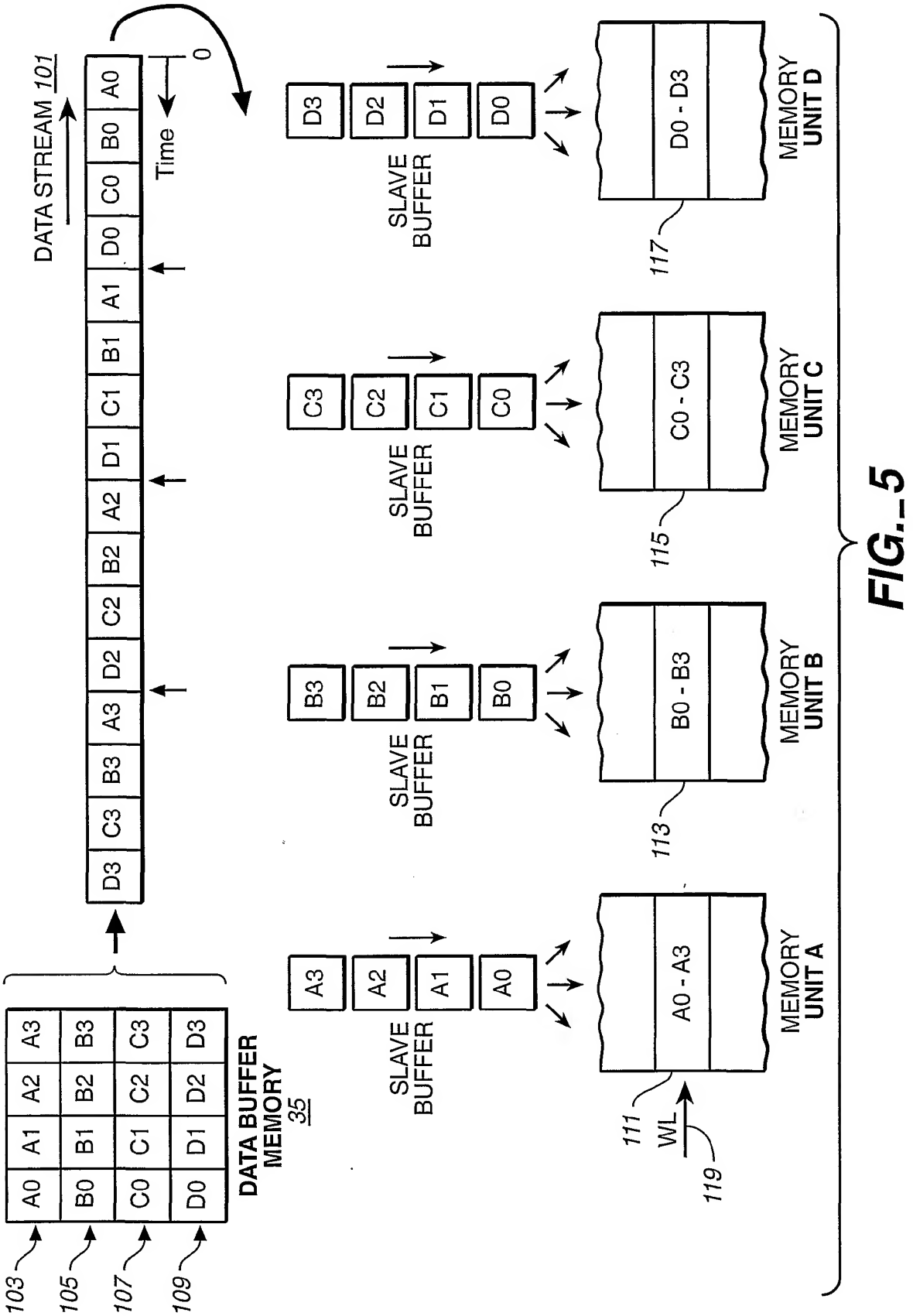

DATA STREAM 101
Time
0
A0 A1 A2 A3
B0 B1 B2 B3
C0 C1 C2 C3
D0 D1 D2 D3
DATA BUFFER MEMORY 35
103
105
107
109
SLAVE BUFFER
A0 A1 A2 A3
B0 B1 B2 B3
C0 C1 C2 C3
D0 D1 D2 D3
MEMORY UNIT A
MEMORY UNIT B
MEMORY UNIT C
MEMORY UNIT D
A0 - A3
B0 - B3
C0 - C3
D0 - D3
111
113
115
117
WL
119

FIG._5

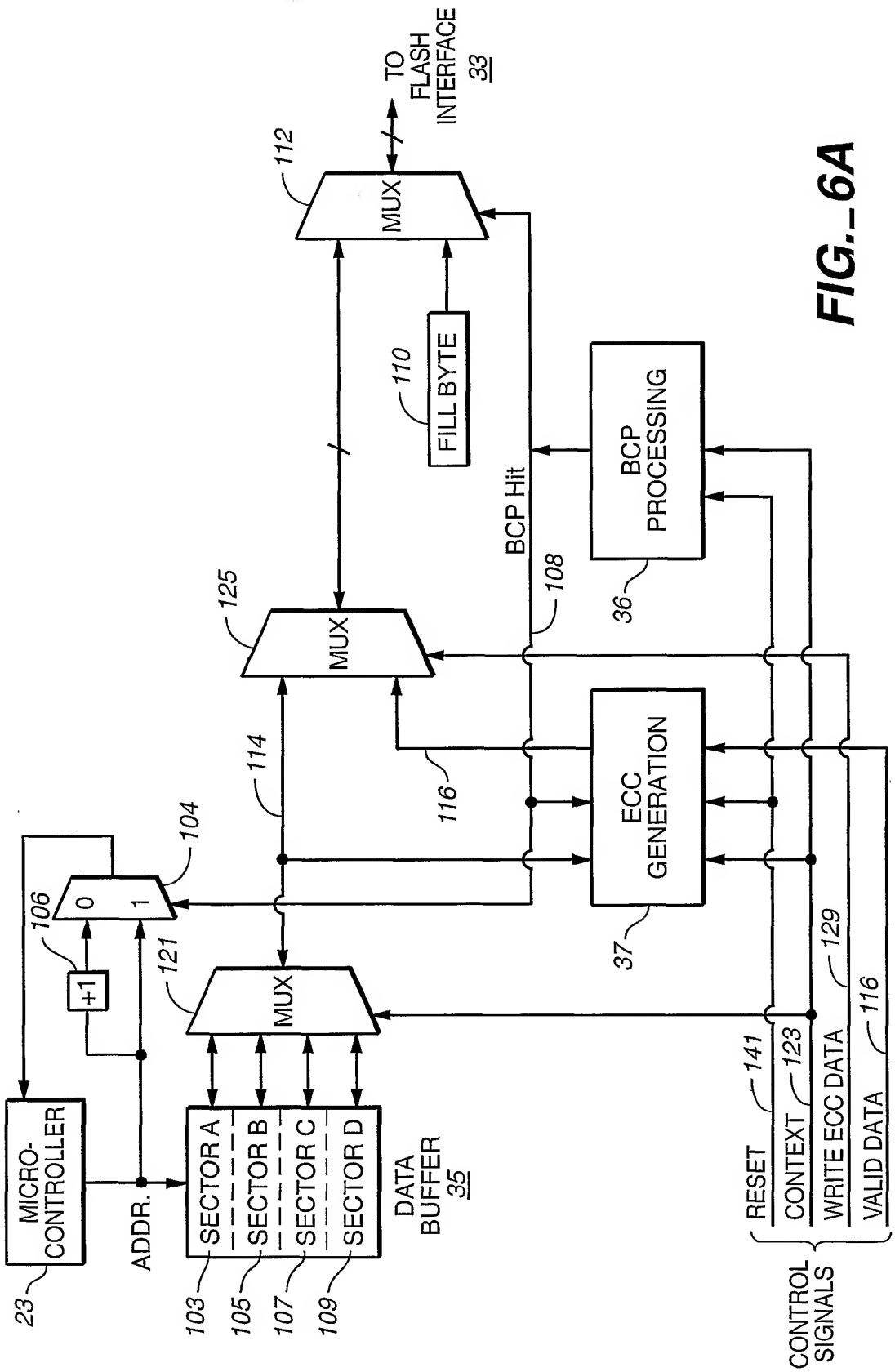

TO FLASH INTERFACE 33
MUX
112
FILL BYTE
110
BCP Hit
BCP PROCESSING
36
108
MUX
125
ECC GENERATION
37
114
116
104
0
1
106
+1
121
MUX
MICRO-CONTROLLER
23
ADDR.
SECTOR A
SECTOR B
SECTOR C
SECTOR D
DATA BUFFER 35
103
105
107
109
RESET
CONTEXT
WRITE ECC DATA
VALID DATA
141
123
129
116
CONTROL SIGNALS

FIG._6A

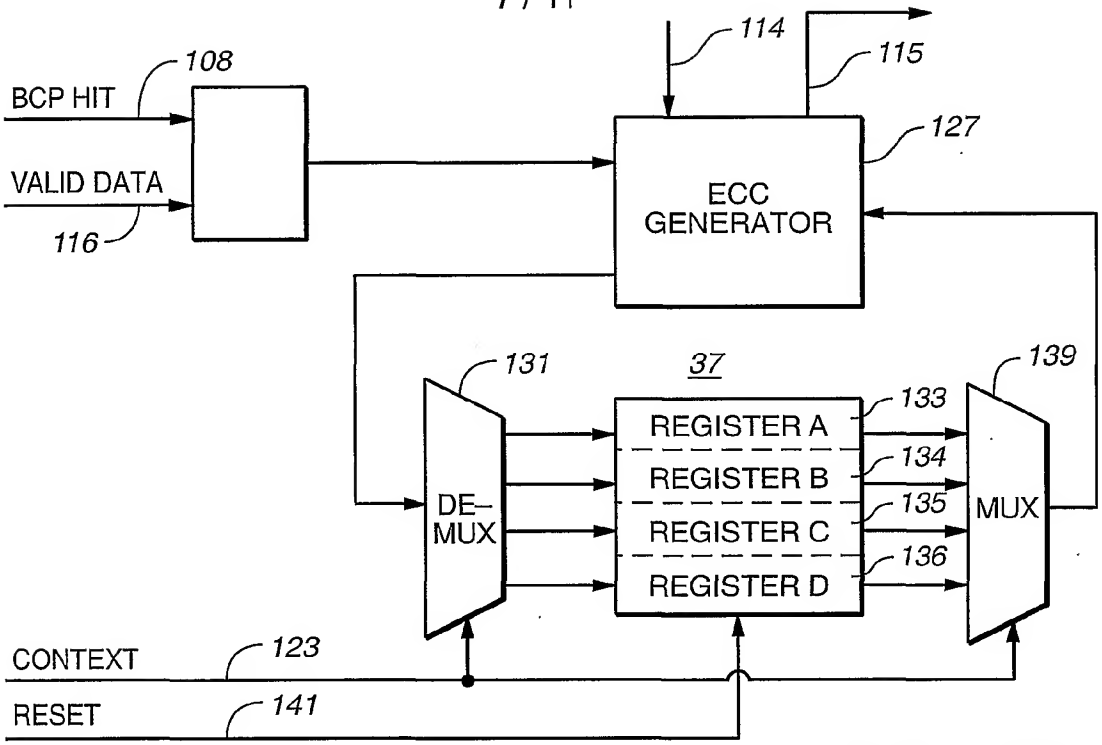

BCP HIT
108
VALID DATA
116
114
115
127
ECC GENERATOR
131
37
139
133
REGISTER A
134
REGISTER B
135
REGISTER C
136
REGISTER D
DE–MUX
MUX
CONTEXT
123
RESET
141

FIG._6B

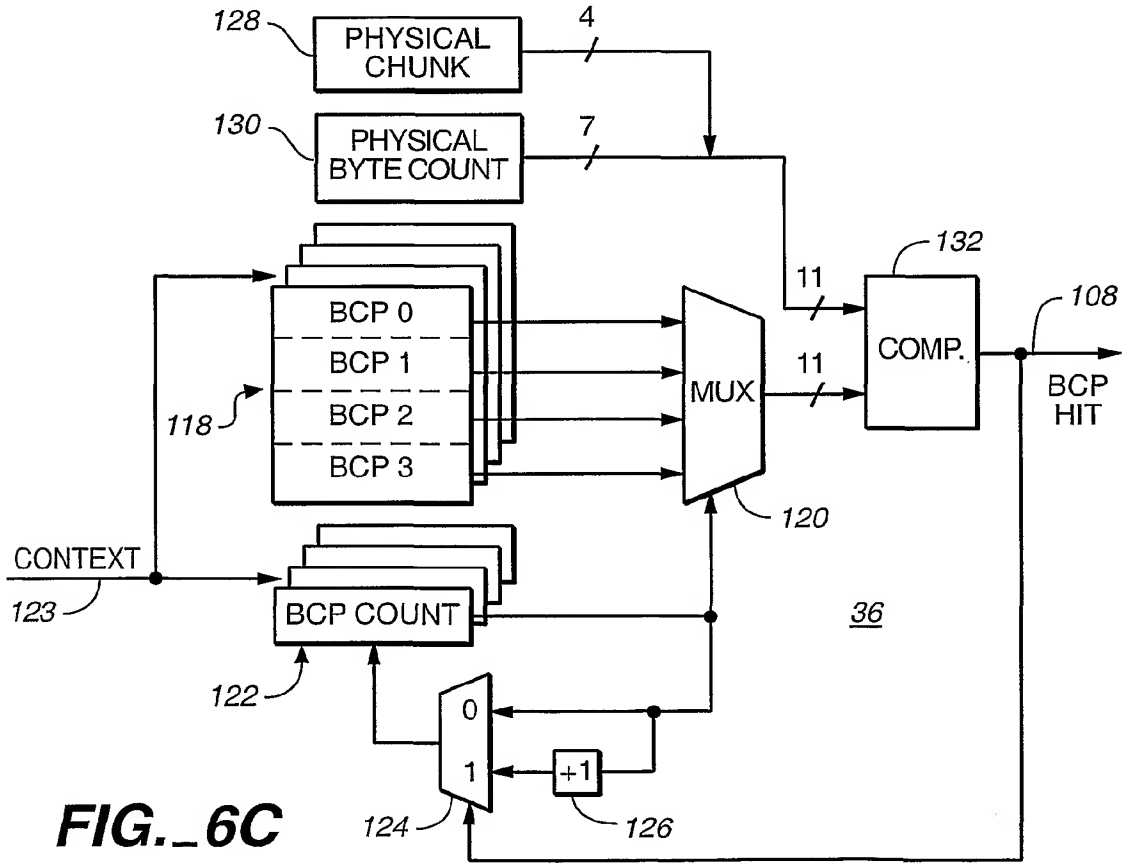

128
PHYSICAL CHUNK
4
130
PHYSICAL BYTE COUNT
7
132
11
108
BCP 0
BCP 1
BCP 2
BCP 3
118
MUX
11
COMP.
BCP HIT
120
CONTEXT
123
BCP COUNT
36
122
0
1
+1
124
126

FIG._6C

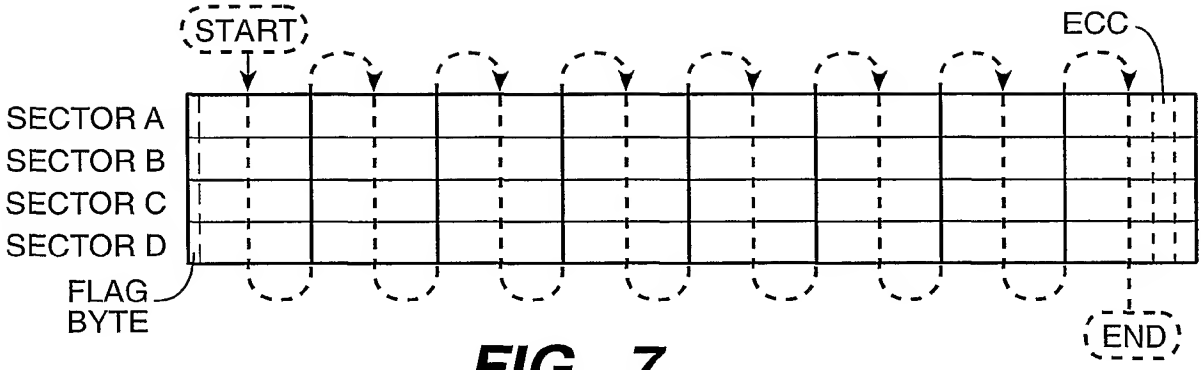


FIG._7

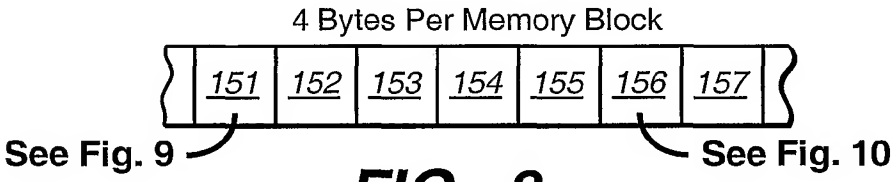


FIG._8

FLAGS Byte 0	VE Byte 1	VPG Byte 2	USAGE Byte 3

FIG._9

FLAGS Byte 0	SPARE UNIT, BLOCK ADDRESS Bytes 1, 2 & 3

FIG._10

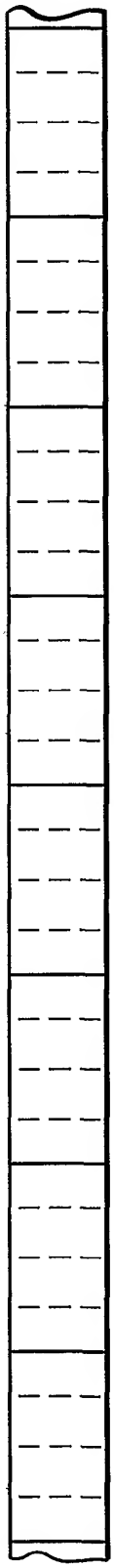

UNIT 0
UNIT 1
UNIT 2
UNIT 3
UNIT 4
UNIT 5
UNIT 6
UNIT 7

FIG._11

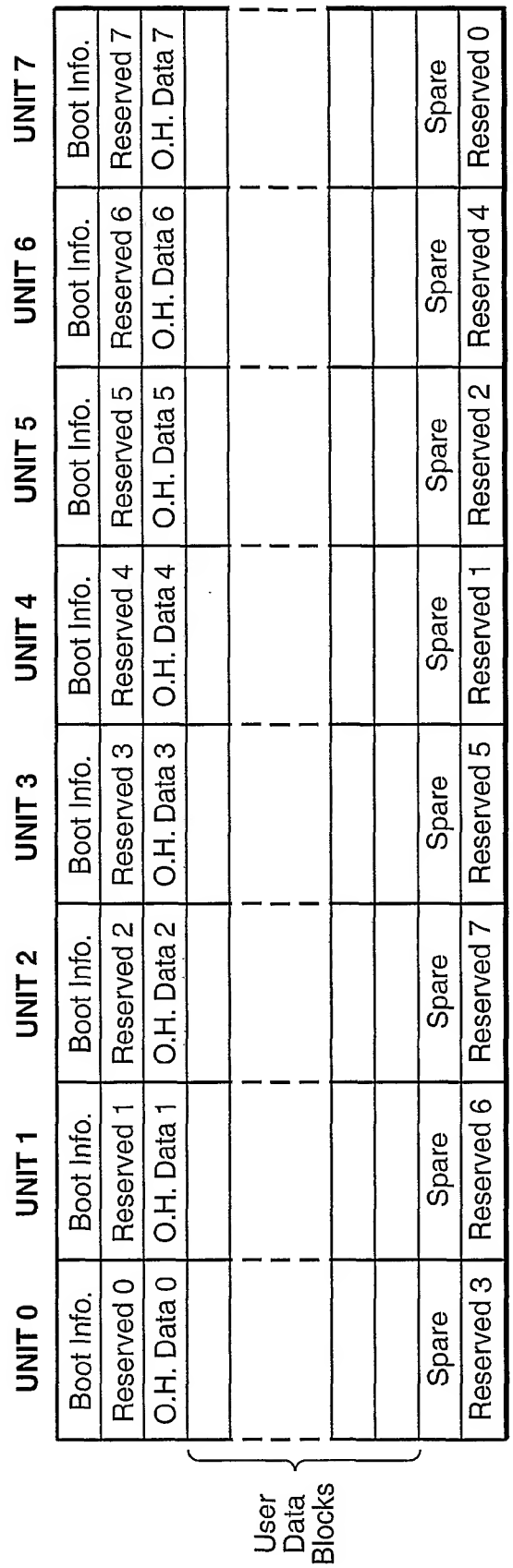

UNIT 0
UNIT 1
UNIT 2
UNIT 3
UNIT 4
UNIT 5
UNIT 6
UNIT 7
Boot Info.
Reserved 0
O.H. Data 0
Spare
Reserved 3
Boot Info.
Reserved 1
O.H. Data 1
Spare
Reserved 6
Boot Info.
Reserved 2
O.H. Data 2
Spare
Reserved 7
Boot Info.
Reserved 3
O.H. Data 3
Spare
Reserved 5
Boot Info.
Reserved 4
O.H. Data 4
Spare
Reserved 1
Boot Info.
Reserved 5
O.H. Data 5
Spare
Reserved 2
Boot Info.
Reserved 6
O.H. Data 6
Spare
Reserved 4
Boot Info.
Reserved 7
O.H. Data 7
Spare
Reserved 0
User Data Blocks

FIG._12

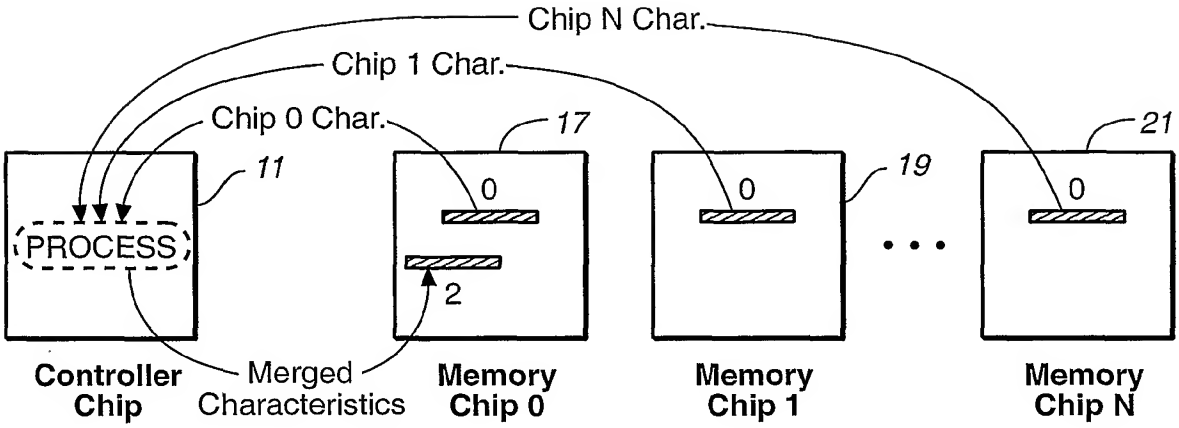


FIG._13

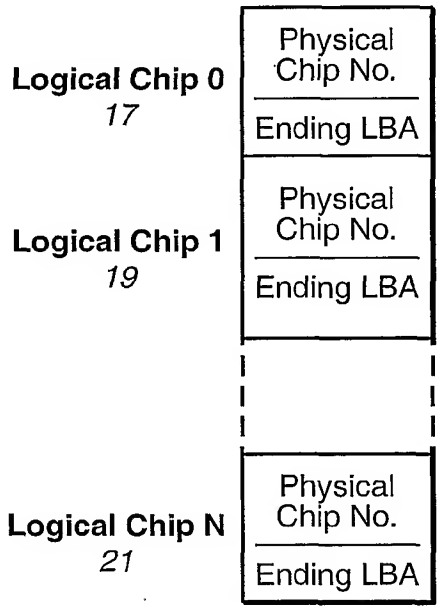


FIG._14

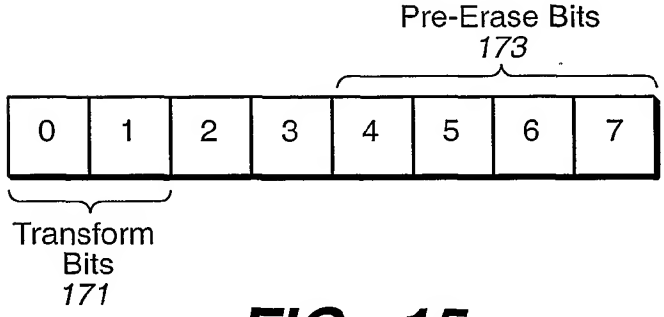


FIG._15

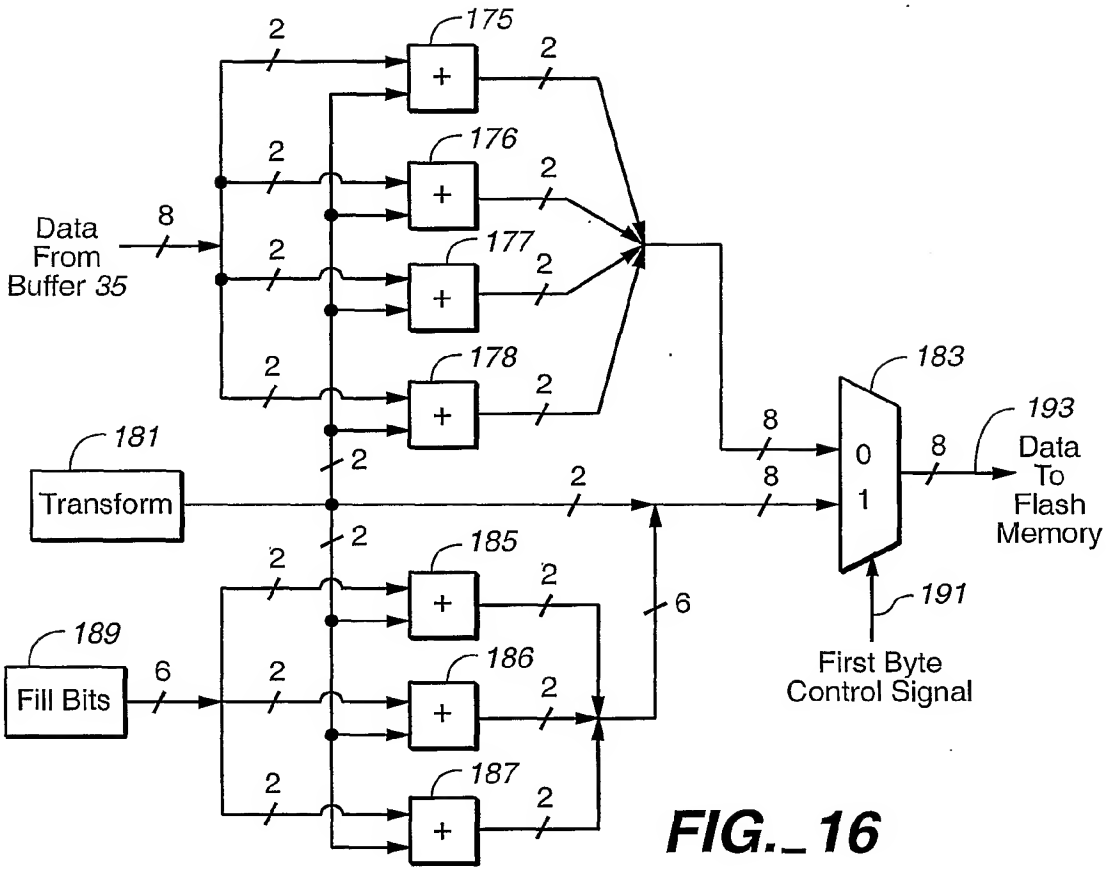

175
176
177
178
183
193
Data To Flash Memory
Data From Buffer 35
181
Transform
185
186
187
189
Fill Bits
191
First Byte Control Signal
0
1
+

FIG._16

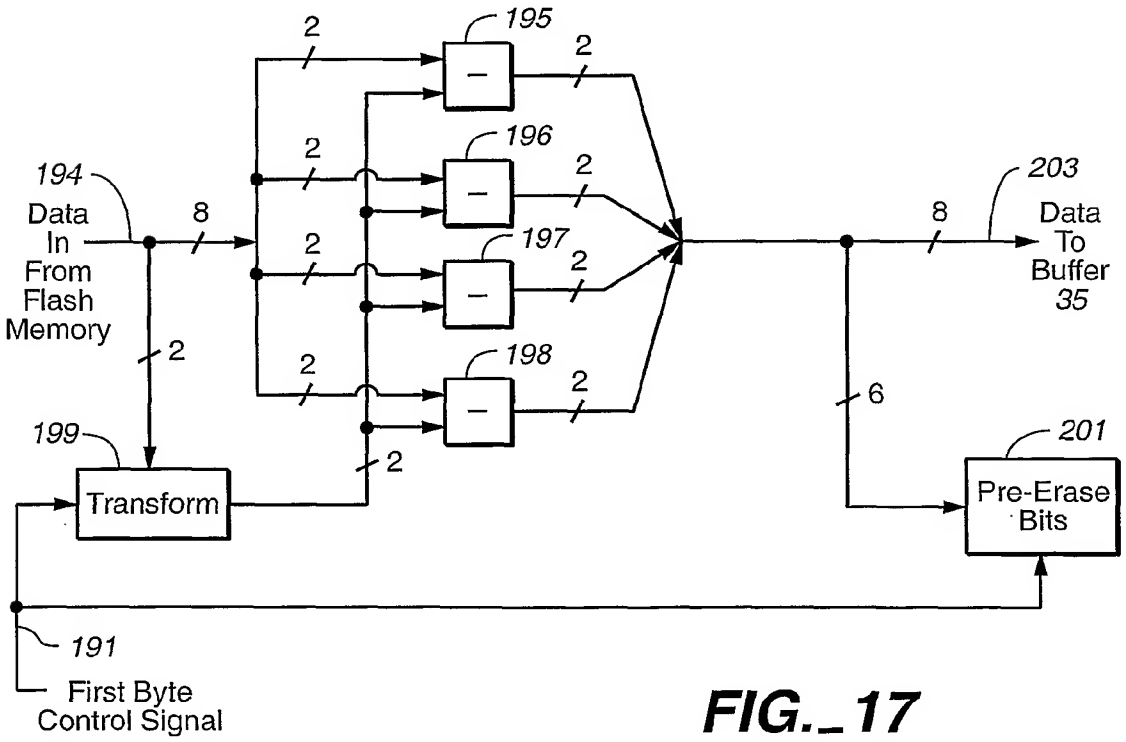

195
196
197
198
194
Data In From Flash Memory
203
Data To Buffer 35
199
Transform
201
Pre-Erase Bits
191
First Byte Control Signal
−

FIG._17